Graph Algorithms and Complexity Aspects on Special Graph Classes

STEWART, ANTHONY,GRAHAM

Durham University

Doctoral Thesis

Graph Algorithms and Complexity Aspects on Special Graph Classes

Author:
Anthony Stewart

Supervisors:
Daniël Paulusma
Matthew Johnson

A thesis submitted for the degree of Doctor of Philosophy

Algorithms and Complexity in Durham
School of Engineering and Computing Sciences

May 2017

Abstract

Graphs are a very flexible tool within mathematics, as such, numerous problems can be solved by formulating them as an instance of a graph. As a result, however, some of the structures found in real world problems may be lost in a more general graph. An example of this is the 4-Colouring problem which, as a graph problem, is NP-complete. However, when a map is converted into a graph, we observe that this graph has structural properties, namely being $\{K_5, K_{3,3}\}$-minor-free which can be exploited and as such there exist algorithms which can find 4-colourings of maps in polynomial time.

This thesis looks at problems which are NP-complete in general and determines the complexity of the problem when various restrictions are placed on the input, both for the purpose of finding tractable solutions for inputs which have certain structures, and to increase our understanding of the point at which a problem becomes NP-complete.

This thesis looks at four problems over four chapters, the first being Parallel Knock-Out. This chapter will show that Parallel Knock-Out can be solved in $\mathcal{O}(n + m)$ time on P_4-free graphs, also known as cographs, however, remains hard on split graphs, a subclass of P_5-free graphs. From this a dichotomy is shown on P_k-free graphs for any fixed integer k.

The second chapter looks at Minimal Disconnected Cut. Along with some smaller results, the main result in this chapter is another dichotomy theorem which states that Minimal Disconnected Cut is polynomial time solvable for 3-connected planar graphs but NP-hard for 2-connected planar graphs.

The third chapter looks at Square Root. Whilst a number of results were found, the work in this thesis focuses on the Square Root problem when restricted to some classes of graphs with low clique number.

The final chapter looks at Surjective H-Colouring. This chapter shows that Surjective H-Colouring is NP-complete, for any fixed, non-loop connected graph H with two reflexive vertices and for any fixed graph H' which can be obtained from H by replacing vertices with true twins. This result enabled us to determine the complexity of Surjective H-Colouring on all fixed graphs H of size at most 4.

Declaration of Authorship

No part of this thesis has previously been submitted for any degree at any institution. Most of the results presented in this thesis have appeared, often in preliminary form, in the papers [41, 44, 57, 58, 60, 61], all of which have been subject to peer review. At the beginning of each chapter it is mentioned where the results presented in that chapter have been published. Although many of the results have been obtained in collaboration, I have been heavily involved in and actively contributed to discussions that led to the results in every section of this thesis.

This work was supported by the Engineering and Physical Sciences Research Council.

Acknowledgements

Seven years ago today, I received my application to study at Durham was accepted (thanks for the memory Facebook). I came to Durham on a 4 year, integrated Masters in Physics. I am now finishing off a Ph.D in Algorithms and Complexity from the Engineering and Computing Sciences department; I think it's fair to say that things haven't exactly gone to plan. Now though I have one of the harder parts of my thesis to write, somehow I have to thank everyone who has got me here in under a page.

I think I first have to thank my parents, who were determined that I would have the best education possible, even if that's not what 11 year old me wanted at the time. The importance they put on my education is the reason I am here today.

I would also like to thank Dani, my wonderful girlfriend who has been a constant source of support (in part because she probably wants me to leave Durham at some point) through the process of writing up this thesis.

Next I must thank my supervisor, Prof. Daniel Paulusma. I first met Daniel in my third year where he was my final year project supervisor, his constant enthusiasm and positivity on my project helped to make it one of the most enjoyable parts of my undergraduate degree and it was him that suggested I carry on with that style of work and do a Ph.D here. Throughout my Ph.D he has been hugely supportive and it has been a pleasure doing research with you.

I would also like to thank everyone I have had the pleasure of with working with, as a co-author or just through discussions, in particular my second supervisor Matthew Johnson and my other co-authors Petr Golovach, Marcin Kaminski, Dieter Kratsch, Barnaby Martin and Dimitrios Thilikos.

Durham has been a huge chapter of my life, and it has taught my so much more than just Physics and Computer Science. I have developed a lot as a person and a significant amount of that is down to the Canoe Club and all of the members over the last 7 years. You have all made my time at Durham such a great experience and got me hooked on a sport I hope I will continue for many year to come.

I could carry on, but to save the margins: to all my friends at Durham and everywhere; to everyone who has helped me to get here; *thank you*!

Contents

List of Theorems

Dedicated to my parents, Ian and Barbara Stewart

"I may not have gone where I intended to go, but I think I have ended up where I intended to be."

Douglas Adams

Introduction

1.1 Graph Theory

1.1.1 A (very) brief introduction

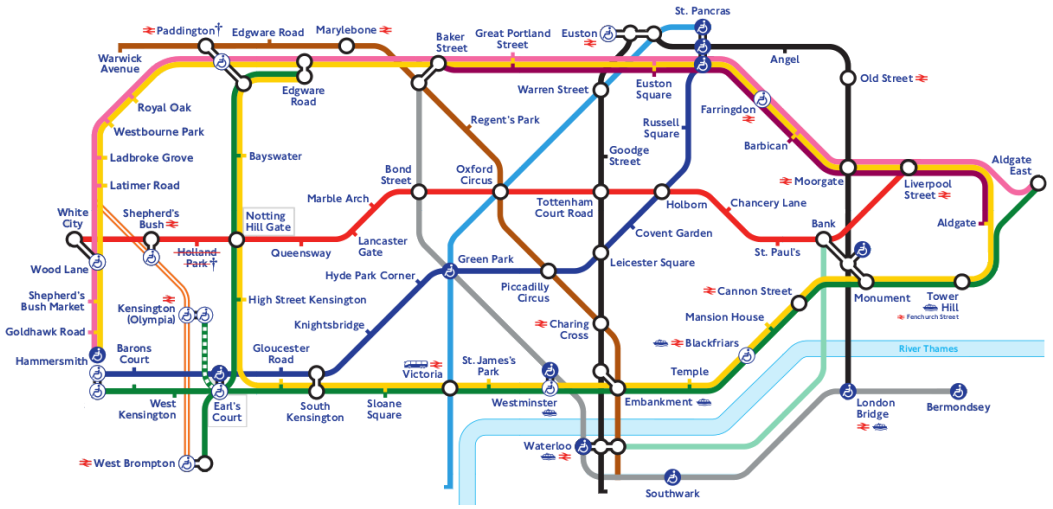


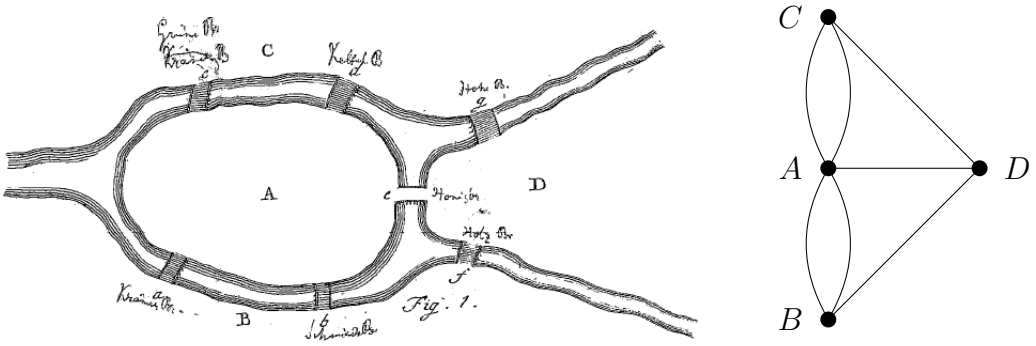
Figure 1.1: The London Tube Map (modified from [74])

A graph is a representation of relationship data; it consists of a set of objects, which we call vertices, and a set of relationships, which we call edges. For example, in Figure 1.1 we see a small part of the well known London Underground map. This is simply a graph where the vertices are stations and the edges form the network of tracks between the stations.

Another common application of graphs is in social networks, where the vertices represent individuals and the edges represent a connection or friendship. Alternatively, graphs could represent circuit diagrams, neighbouring regions, or a multitude of other scenarios.

Put simply, *Graph Theory* is a branch of Discrete Mathematics which looks at problems on data which can be formulated as a set of relationships.

1.1.2 The Seven Bridges of Königsberg



(A) Diagram by Leonhard Euler [30] (B) As a graph G.

FIGURE 1.2: The Seven Bridges of Königsberg.

The Seven Bridges of Königsberg is a well known problem presented in a paper by Leonhard Euler [30] which was later translated into English in several books including *Graph Theory, 1736-1936* by Biggs, Lloyd and Wilson [5]. Königsberg was a city split by the river Pregel, with four distinct bodies of land, which we shall call regions, which were connected by 7 bridges as depicted by Figure 1.2a.

The problem is to formulate a route through the city, navigating the river using only the bridges and using each bridge exactly once. Euler used a branch of mathematics known as *geometry of position* (*geometriam situs*) [30]; this branch of mathematics looks at the position of objects, but not the distances between them.

Euler first mentions that the problem could be solved by looking at all possible paths, but dismissed this idea as "laborious" and "impossible for configurations with significantly more bridges" [5]. We note in Figure 1.2a, that Euler has labelled the land masses A to D and the bridges a to g.

To solve this problem, Euler calculated the number of times a region would have to be visited in a valid route. The result he found was that if there are more than two regions (or exactly one region) to which an odd number of bridges lead, it is impossible to create a tour. This can be seen by considering the fact that to visit

a region, we must enter by one bridge and leave by another, hence two bridges are required. Visiting the region again requires a further two bridges. The only exceptions are the regions from which we start and end, since only one bridge is required to leave or arrive upon them respectively. Alternatively, if we start and end at the same region, then we see that all regions must have an even number of bridges leading to them.

We can formulate Euler's result using graph theoretic terminology. We use a vertex to represent each region; a bridge between two regions is represented by an edge between the two corresponding vertices. We see a depiction of this graph in Figure 1.2b.

We call the number of edges ending at a vertex the degree of the vertex. For example, the middle-left vertex (A) in Figure 1.2b has degree 5, whereas all of the other vertices have degree 3. For a valid walk across the bridges, Euler's result states that there is a valid solution only if either exactly 0 or exactly 2 vertices have odd degree. Since all four vertices have odd degree, the Seven Bridges of Königsberg problem has no solution.

Whilst Euler did not explicitly use graph theory in his proof, his work is widely regarded to be the earliest example of this branch of Mathematics [5].

1.1.3 Terminology

In this thesis we mostly use the notation presented in *Graph Theory* by Reinhard Diestel [26]. Some notation which is specific to each chapter shall be defined at the start of the relevant sections, however, the more general notations shall be discussed here.

We denote a set as a comma-separated list within braces, for example we may define the set $X = \{x, y, z\}$.

For clarity, we sometimes use an ellipsis to simplify increasing sequences; for example $\{x_1, \ldots, x_4\} = \{x_1, x_2, x_3, x_4\}$ and $1 + \cdots + n = \sum_{i=1}^{n}(i)$.

We now present a table of the commonly used set notations:

Not'	Usage	Meaning
$\in$	$x \in X$	x is in the set X
$\subseteq$	$Y \subseteq X$	every element of Y is in X
$=$	$Y = X$	X and Y contain the same elements
$/$	$Y \neq X$, $x \notin X$	Y does not equal X, x is not in the set X
$\subset$	$Y \subset X$	$Y \subseteq X$ and $Y \neq X$
$\supseteq$	$Y \supseteq X$	every element of X is in Y
$\supset$	$Y \supset X$	$Y \supseteq X$ and $Y \neq X$
$\wedge$	$\Phi_1(x) \wedge \Phi_2(x)$	True if and only if $\Phi_1(x)$ and $\Phi_2(x)$ are true
$\vee$	$\Phi_1(x) \vee \Phi_2(x)$	True if and only if $\Phi_1(x)$ or $\Phi_2(x)$ are true
$\mid$	$\{x \mid \Phi(x)\}$	Set built on x such that function $\Phi(x)$ holds
	$\{x \in X \mid \Phi(x)\}$	Shorthand for $\{x \mid x \in X \wedge \Phi(x)\}$
$\cup$	$S \cup T$	The set of elements which belong to either S or T
$\cap$	$S \cap T$	The set of elements which belong to both S and T
$\setminus$	$S \setminus T$	The set of elements which belong to S but not T
$+$	$S + x$	The set S with the addition of x $(S \cup \{x\})$
$-$	$S - x$	The set S with x removed $(\{s \in S \mid s \neq x\})$
$\forall$	$\forall x \in X, \ldots$	For every x in X, the following is true...
$\exists$	$\exists x \in X, \ldots$	For at least one x in the set X, the following is true...
$\bigcup$	$\bigcup_{i=1}^{n}(S_i)$	The union of all sets S_1 to S_n : $S_1 \cup \ldots \cup S_n$
	$\bigcup_{S \in X}(S)$	The union of all sets within X.
$\bigcap$		As defined for $\bigcup$ but with the intersection.
$\|\|$	$\|S\|$	The number of elements in the set S

Table 1.1: Basic set notation

We define a graph as a tuple or pair of sets, $G = (V, E)$. Each element of V is a vertex and each element of E is an edge, an unordered pair of vertices from the set V. We denote vertices by lower-case letters, usually later in the alphabet for example u or v. We also denote edges by lower-case letters, usually earlier in the alphabet, e.g. $e = uv$ where uv is shorthand for $\{u, v\}$.

We generally use lower-case letters from the middle of the alphabet to denote other values. Where this can be assumed without ambiguity, n is the number of vertices in a graph and m is the number of edges.

Where we have two or more graphs, for example, G and H, we may define them as follows: $G = (V_G, E_G)$ and $H = (V_H, E_H)$ so as to be able to distinguish between the vertex sets V_G and V_H and between the edge sets E_G and E_H.

A *function* is a mapping from elements of one set to elements of another set. For example we would write $f : V_G \to V_H$ to denote a function which takes a vertex from a graph G and returns a vertex from a graph H. In such a case we call V_G the *domain* for f and we would call V_H the *codomain* of f. We define the *image* of a function as the set of values which are mapped to by at least one element in the domain.

In general, given a function $f : X \to Y$ and a value $x \in X$, we denote the value to which x is mapped by the function f as $f(x)$. For a set $S \subseteq X$, we use the notation $f(S)$ to denote the set $\{f(x) \mid x \in S\}$.

A function is *injective* if every element in the codomain is mapped to by at most one value in the domain and a function is *surjective* if every value in the codomain is mapped to by at least one value in the domain, in other words if the codomain and image are the same. A function is *bijective* if it is both injective and surjective, that is if the domain and codomain are of the same size and every element of the codomain is mapped to by exactly one element of the domain.

1.1.4 Graph Classes

With an undirected graph, edges are not directional, that is if u is adjacent to v, then v is by definition adjacent to u and we consider two edges $uv \in E$ and $vu \in E$ to be indistinguishable. Conversely a directed graph $G = (V, A)$ (also known as a digraph) has a set of *arrows* which are defined with a direction, such that $uv \in A$ and $vu \in A$ represent different things. It may be the case that while there is an arrow from u to v, the reverse is not true.

We say that a graph has *multiple edges* if it contains at least two edges with the same endpoints. We call an edge which starts and ends at the same vertex, $e = vv$, a self-loop. We say a graph is simple if it does not contain multiple edges or self-loops; unless otherwise specified, all graphs in this thesis are simple and undirected.

We say that a graph $H = (V_H, E_H)$ is a *subgraph* of $G = (V_G, E_G)$ if V_H is a subset of V_G and E_H is a subset of E_G. We say that H is a *spanning* subgraph if $V_H = V_G$. For a subset $S \subseteq V$, we let $G[S]$ denote the subgraph of G *induced* by S, which has vertex set S and edge set $\{uv \in E_G \mid u, v \in S\}$.

A set $I \subseteq V_G$ is called an *independent set* of G if no two vertices in I are adjacent to each other in G. A subset $C \subseteq V_G$ is called a *clique* of G if every pair of vertices in C are adjacent to each other in G. We say that a subset with a property is *maximal*, if there is no vertex which can be added to the subset such that the new subset also has the property. For example a subset $C \subseteq V_G$ is called a *maximal clique* of G if C is a clique and there is no vertex outside of C adjacent to every vertex within C.

A subset $D \subseteq V_G$ is a *dominating set* of a graph $G = (V, E)$ if every vertex of G is in D or adjacent to a vertex in D. We say a vertex v is a *dominating vertex* if $\{v\}$ is a dominating set. We say a graph is disconnected if we can partition it into two components such that there are no edges between the two components. A subset

$S \subseteq V_G$ is called a *cutset* or *cut* if the removal of the vertices in S disconnects the graph. That is, if the graph $G[V_G \setminus S]$ is disconnected.

The *union* of two graphs G and H is the graph with vertex set $V_G \cup V_H$ and edge set $E_G \cup E_H$. If $V_G \cap V_H = \emptyset$, then we say that the union of G and H is *disjoint* and write $G + H$. We denote the disjoint union of r copies of G by rG.

We say that two graphs G and H are isomorphic if they are structurally the same. That is, there exists a bijective function $f : V_G \to V_H$ such that uv is an edge in G if and only if $f(u)f(v)$ is an edge in H.

Using these properties, we can introduce the concept of graph classes. A class of graphs is a subset of the set of all possible graphs, in which all graphs share certain properties or structures.

For $n \geq 1$, the graph P_n denotes the *path* on n vertices, that is the graph with $V_{P_n} = \{u_1, \ldots, u_n\}$ and $E_{P_n} = \{u_i u_{i+1} \mid 1 \leq i \leq n-1\}$. When we refer to a path of length m, we are referring to a path with m edges which has $m+1$ vertices, we denote this P_{m+1}. A *cycle* is similar to a path, except that the end vertices are also adjacent such that the graph forms a circle. For $n \geq 3$, the graph C_n denotes the *cycle* on n vertices, that is, $V_{C_n} = \{u_1, \ldots, u_n\}$ and $E_{C_n} = \{u_i u_{i+1} \mid 1 \leq i \leq n-1\} \cup \{u_n u_1\}$.

The graph K_n denotes the *complete graph* on n vertices, that is, the n-vertex graph whose vertex set is a clique. A graph is *complete bipartite* if its vertex set can be partitioned into two sets, such that two vertices u and v are adjacent if and only if u and v belong to different partitions. The graph $K_{p,q}$ is the complete bipartite graph with partitions of sizes p and q, respectively.

We may also describe a graph class by forbidding certain structures. Let G be a graph and let $\{H_1, \ldots, H_p\}$ be a set of p graphs. We say that a graph G is $(H_1, \ldots, H_p)$-free if G has no induced subgraph isomorphic to a graph in $\{H_1, \ldots, H_p\}$. If $p = 1$ we may write H_1-free instead of (H_1)-free. For example a

graph which is K_3-free has no cliques of size larger than 2. That is, if a vertex u is adjacent to v and v is adjacent to w, it is not possible for u to be adjacent to w.

As well as forbidding certain induced subgraphs we can also describe structures in other ways. An *edge contraction* is an operation where given an edge $uv \in E_G$, we remove the vertices u and v. We then insert a new vertex which is adjacent to every vertex which was previously adjacent to u or v; this operation does not create a self-loop. An example of this is shown in Figure 1.3.

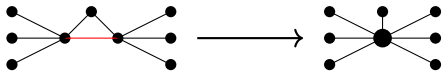


FIGURE 1.3: An example of an edge contraction.

Graph minors are commonly used to describe forbidden structures. A graph H is a *minor* of a graph G if we can get to H from G by deleting vertices and deleting or contracting edges. We say that a graph G is $(H_1, \ldots, H_p)$-minor-free if none of the graphs $H_1, \ldots, H_p$ are minors of G. As with forbidden induced subgraphs, if $p = 1$ we may write H_1-minor-free instead of (H_1)-minor-free.

A commonly used graph glass is that of *planar* graphs. A graph is planar if it can be drawn on the surface of a sphere without any edges crossing. We call such a drawing a *planar embedding*. It has been shown that the class of planar graphs is the same as the class of $(K_5, K_{3,3})$-minor-free graphs [25]. A related class is that of *outerplanar* graphs which are what we call a *subclass* of planar graphs. That is every outerplanar graph is also a planar graph, although not every planar graph is outerplanar. Conversely we say that planar graphs are a *superclass* of outerplanar graphs. A graph is outerplanar if it can be drawn in the plane such that no vertex is entirely surrounded by edges.

We say that a graph H is a *k-factor* of a graph G if H is a *k-regular* spanning subgraph of G. By k-regular we mean that every vertex has degree k. A 1-factor of a graph is a disjoint union of paths of length 1. A graph is *k-factorable* if it admits a k-factor.

We say that H is an $[i, j]$-*factor* of a graph G if H is a spanning subgraph of G and each component of H is either i-regular or j-regular. A $[1, 2]$-factor of a graph is a disjoint union of cycles and paths of length 1.

In this thesis we also make reference to a similar concept which we define as an (i, j)-*factor*. We say that H is an (i, j)-factor of a graph G if H is a spanning subgraph of G and H can be separated into two components such that every vertex in one component has degree at most i and every vertex in the other has degree at most j. A $(1, 2)$-factor of a graph is a disjoint union of paths of length at most 2.

There exist many more graph classes, far too many to name in this section so we recommend Information System on Graph Classes and their Inclusions at graphclasses.org [25] as a reference.

1.1.5 Graph Parameters

We use certain graph parameters to describe aspects of a graph. The simplest example of a parameter is simply the *size* of a graph which is the number of vertices in the graph. We denote this $|G| = |V_G|$. Another example of a parameter is the *clique number* of a graph which is the size of the largest clique in a graph. In general a parameter is a function which maps a graph to a number. A commonly used parameter is the treewidth of a graph, to understand this we must first explain a tree decomposition of a graph.

A *tree decomposition* of a graph G is a pair (T, X) in which T is a tree and $X = \{X_i \mid i \in V_T\}$ is a collection of subsets (called *bags*) of V_G such that the following three conditions hold:

i) $\bigcup_{i \in V_T} X_i = V_G$,

ii) for each edge $xy \in E_G$, $x, y \in X_i$ for some $i \in V_T$, and

iii) for each $x \in V_G$ the set $\{i \mid x \in X_i\}$ induces a connected subtree of T.

The *width* of a tree decomposition $(\{X_i \mid i \in V_T\}, T)$ is equal to one less than the size of the largest bag in X. The *treewidth* $\mathbf{tw}(G)$ of a graph G is the minimum width over all tree decompositions of G. If T is restricted to be a path, then we say that (X, T) is a *path decomposition* of a graph G. Using our general description of a parameter, we can describe treewidth as a function $\mathbf{tw} : \mathcal{G} \to \mathbb{Z}$ where $\mathcal{G}$ is the set of all graphs. The *pathwidth* $\mathbf{pw}(G)$ of G is the minimum width over all path decompositions of G. A class of graphs $\mathcal{G}$ has *bounded* treewidth (pathwidth) if there exists a constant p such that the treewidth (pathwidth) of every graph from $\mathcal{G}$ is at most p. Treewidth is a very useful concept since many problems can be solved easily on trees since no two vertices in different branches are adjacent. Graphs with bounded treewidth have a treelike structure which often allows us to use similar approaches to solve problems.

We will define any additional graph parameters that are required as they are needed.

1.2 Complexity Theory

1.2.1 Running Time of an Algorithm

Some problems are harder than others; a large part of modern security is based on the fact that it is relatively easy to multiply large numbers together but there is no known algorithm to quickly find the factors of a large number [87].

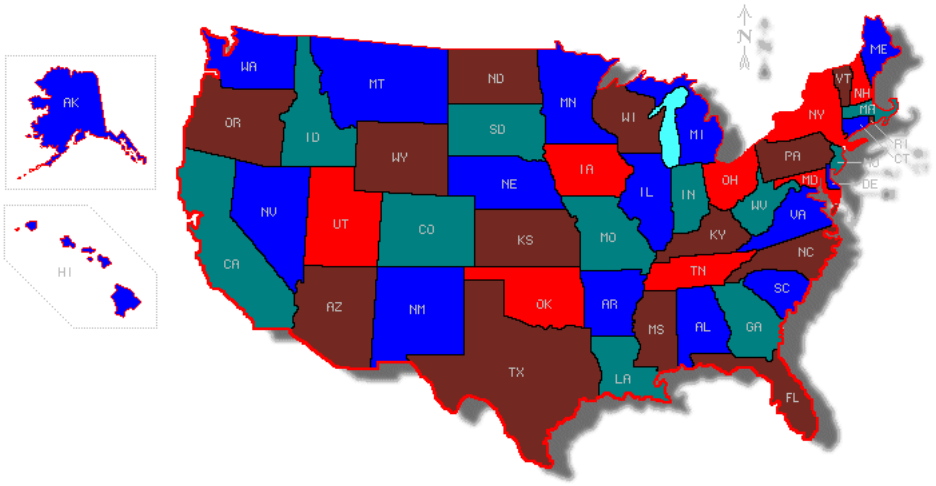


FIGURE 1.4: The States of America coloured with four colours.

A well studied problem is that of colouring. Given a map with separate regions, can you colour each region so that no two regions that share a border are the same colour? In Figure 1.4 we see the United States of America coloured using four colours.

We study this problem from a graph theoretic perspective. First we describe the problem instance as a graph where each vertex is a region and two vertices are adjacent if and only if they share a border. We call the problem of labelling the vertices of a graph with k distinct colours such that no two adjacent vertices are assigned the same colour k-COLOURING.

If the problem were to find a colouring if one exists, using only two colours, then we can do that with relative ease. We now present a simple algorithm for 2-Colouring.

2-Colouring

input : a connected graph G

output : a valid *2-colouring* if one exists; `no` otherwise

Step 1. Choose any vertex and colour it *red*.

Step 2. Repeat the following step until either every region is coloured (in which case return the colouring) or one region is given a colour which is the same as one of its neighbours (in which case return `no`):

> For a region which has just been coloured *red* or *blue*, colour each of its neighbours *blue* or *red* respectively.

As a Mathematician, however, simply stating "it's easy" is not enough; we want to quantify exactly how easy it is. To do this we introduce a concept we call *big $\mathcal{O}$ notation*. Big $\mathcal{O}$ notation is used to describe the limiting behaviour of a function as the argument tends towards infinity. In computer science we use big $\mathcal{O}$ notation to classify algorithms based up how a limited resource, usually time or memory, scales with the size of the input. Within this thesis we concerned with how much time an algorithm requires to complete.

We say that an algorithm is $\mathcal{O}(f(n))$ if there exist constants $k, n_0 > 0$ such that in the worst case, the algorithm completes in a time bounded by $k \cdot f(n)$ for all inputs of size $n > n_0$. We call this the *time complexity* of the algorithm. Since we are dealing with the asymptotic limits, we can neglect smaller order terms, for example $\mathcal{O}(n^2 + n) = \mathcal{O}(n^2)$ and $\mathcal{O}(2^n + n^2) = \mathcal{O}(2^n)$.

We can determine the complexity for the 2-Colouring algorithm described above on a map with n regions. We can assume that it takes a constant time to colour a region and $\mathcal{O}(n)$ time to find all regions adjacent to another region. Since we must do the latter for each region, we see that the entire algorithm runs in $\mathcal{O}(n^2)$. We say that the running time is polynomial with respect to the size of the input since the running time can be bound by a polynomial function.

In practice we say a problem is *tractable* if there exists an algorithm that can solve it in polynomial time.

1.2.2 Non-Deterministic Polynomial Time

We say that a problem is a *decision problem* if it has a `Yes`/`No` answer. For example the corresponding decision problem for k-Colouring asks: Can we colour a graph G using only k colours? To which the answer would be either `Yes` or `No`.

We will now consider the colouring problem again, but this time we want to know if it is possible to colour a map using three colours. If we start as we did before, we colour a vertex *red*, then we colour one of its neighbours *blue* and any neighbours of both of these vertices *green*. Next we choose a vertex adjacent to only one of these vertices, without loss of generality we assume that its neighbour which has already been coloured is *green*. We now have to decide whether to colour this new vertex *red* or *blue*. We see an example of this in Figure 1.5.

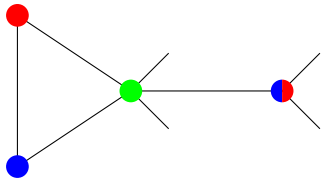


Figure 1.5: An example of a partial 3-Colouring on a subset of a graph.

We cannot assign a colour to this fourth vertex since it could be either *red* or *blue* and a solution may be possible with one colour assigned but not the other.

Computers are *deterministic*, that means that in a given state we can do one operation, for example we can set the fourth vertex to either *red* or *blue*. Or we can create an additional instance of the problem and try and solve the problem twice, once with the vertex set to *red* and a second time with the vertex set to *blue*. Obviously this would double the running time each split and as such would give us an algorithm which ran in $\mathcal{O}(2^n)$ (exponential time).

We can, however, consider a hypothetical "non-deterministic" computer which could set the fourth vertex to both *red* and *blue* simultaneously and then could proceed like this in the same way as the algorithm for 2-Colouring. We say that problems which could be solved in polynomial time on a hypothetical non-deterministic machine are *non-deterministic, polynomial time*. We define a class, NP, which is the set of all decision problems which can be solved in polynomial time on a non-deterministic machine and we say that such problems are "in NP".

We denote the class of all decision problems which can be solved in polynomial time on a traditional, deterministic, machine as P. Since a non-deterministic machine can do everything that a deterministic machine can do, it is clear that every problem in P is also in NP. Further, it has been shown [1] that a problem belongs to NP, if and only if we can verify in polynomial time (with respect to the size of the problem instance) whether or not a candidate solution to it is valid. For example, if we were given a 3-colouring of a graph, we can simply check each pair of vertices ($\mathcal{O}(n^2)$) and check that no two adjacent vertices have the same colour. As such we see that the 3-Colouring problem is in NP.

We have seen above that we cannot easily adapt our 2-Colouring algorithm to solve 3-Colouring in polynomial time. In fact, despite a huge amount of research on the area, no one has managed to find an algorithm which can solve 3-Colouring in polynomial time, nor has anyone successfully proved that such

an algorithm does not exist. The best known algorithm for 3-Colouring is $\mathcal{O}(1.3289^n)$ which was discovered by Beigel and Eppstein in 2005 [4].

In fact, there is no problem within NP which has been proven not to have a polynomial time algorithm which leads to the following question:

Does P = NP?

This is arguably the biggest question in Computer Science and one of the Clay Mathematics Institutes Millennium Prize Problems. It is widely regarded as the case that there are probably problems in NP with no polynomial time solution, that is $\mathsf{P} \subsetneq \mathsf{NP}$.

1.2.3 Complexity Classes

We have already established our first two complexity classes, P and NP. From this we can define two further classes, the first of which is called NP-hard. Let $P : \mathcal{G} \to \{\texttt{Yes}, \texttt{No}\}$ and $Q : \mathcal{H} \to \{\texttt{Yes}, \texttt{No}\}$ be decision problems. We say that P can be reduced to Q if there exists some function $r : \mathcal{G} \to \mathcal{H}$ which runs in polynomial time such that $P(i) = Q(r(i))$. That is, given an instance of P, we can create an instance of Q such that an answer to the latter problem is an answer to the former.

We say that a problem Q is NP-hard if every problem in NP can be reduced to Q in polynomial time. Hence if Q were to have a polynomial time algorithm then all of NP would as well and we would see that P = NP. We define the class of NP-complete problems as those problems which are both in NP and NP-hard. We see an example of this in Figure 1.6.

Throughout this thesis we will assume that $\mathsf{P} \subsetneq \mathsf{NP}$; we will be looking at problems in NP and trying to either provide a polynomial time algorithm to solve them or prove that they are NP-complete.

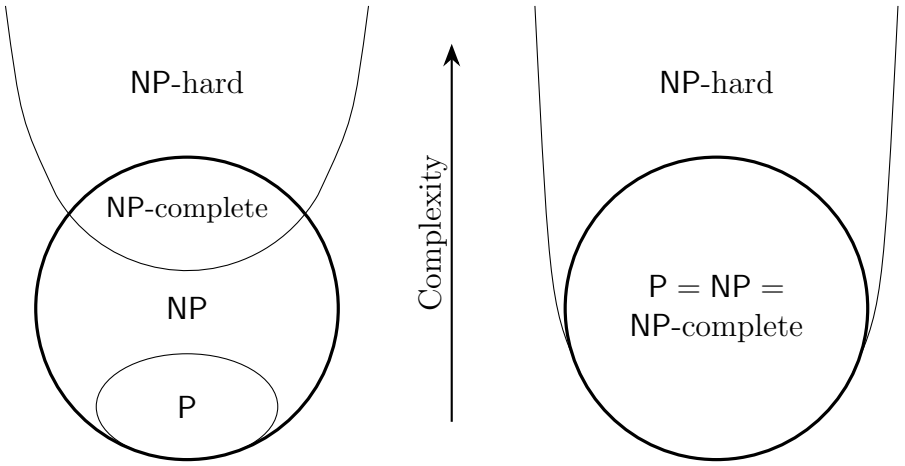


FIGURE 1.6: Euler diagram for P, NP, NP-complete and NP-hard sets of problems [28]. The scenario where $\mathsf{P} \subsetneq \mathsf{NP}$ is shown on the left, whereas the scenario where $\mathsf{P} = \mathsf{NP}$ is shown on the right.

A further complexity class is called FPT. A problem is *fixed parameter tractable* if it has an input of the form (x, k) where k is some parameter, which is relatively small compared to x, and its running time is in $\mathcal{O}\left(\mathrm{f}(k)\,\mathrm{p}(|x|)\right)$ where p is a polynomial function on x and f is some function that depends only on k [27].

1.2.4 Coping with Hardness

In an ideal world, when presented with a problem, we would like a polynomial time algorithm which will give us the optimal solution for every possible instance of the problem. As soon as it becomes clear that the problem is NP-hard, it is clear that if we want a polynomial time algorithm, we have to make concessions somewhere. One branch of Computer Science in which there is extensive research is approximation algorithms. These algorithms may not always find the best solution, or they may have a small probability of returning the wrong answer. However, in many situations these algorithms will provide a reasonable solution.

This thesis looks at the other approach, seeing if there are tractable solutions when we apply restrictions on the input to the problem. It is often the case that real world inputs have structures within the data which can be exploited to achieve

faster running times. These structures may not always be present in the general version of the problem. We shall recall the colouring problem just once more in this chapter. The problem of 4-Colouring is NP-complete [40]. However, maps have a property which is not always present in graphs which we can make use of; we can draw any map (as long as it does not contain any disconnected regions) as a graph with no two edges crossing over. We call this type of graph a *planar* graph and as previously mentioned, it can be described using forbidden minors.

In 1852 [75] Francis Guthrie, while trying to colour the map of counties of England, noticed that only four different colours were required to colour any map he considered. He conjectured that any map could be coloured using only four colours. There were several early attempts to prove this conjecture [93] but these were all proven false.

During the 1960s and 1970s, a German mathematician named Heinrich Heesch started developing methods to solve this problem using computers. Unfortunately he was unable to secure the time on a supercomputer required to finish his research [100], however, others took up his methods and approach. In 1976, it was proven by Appel and Haken [2]. Since then there have been algorithms found which can find a 4-colouring in $\mathcal{O}(n^2)$ [90].

In this thesis we will study four problems; Parallel Knock-Out (Chapter 2), Minimal Disconnected Cut (Chapter 3), Square Root (Chapter 4) and Surjective H-Colouring (Chapter 5) which are all NP-Complete in general. Each chapter looks at the complexity of the problem with restricted graph classes.

Chapter 2

Parallel Knock-Out on P_k-free Graphs

Lampert and Slater [65] introduced the following procedure, called a *knock-out scheme*, for a graph $G = (V, E)$ with minimum degree at least 1:

1. Let **every** $v \in V$ select exactly one neighbour.
2. Eliminate all selected vertices.
3. Repeat the procedure with the smaller graph on the remaining vertices.
4. Stop as soon as
   a. there are no vertices left, or
   b. one of the remaining vertices has degree 0.

If there exists a scheme which stops due to step 4a, then we say that the graph G is *KO-reducible*. Steps 1 and 2 form a *round*.

The Parallel Knock-Out problem is to decide whether or not a graph G is KO-reducible. This problem is known to be NP-complete in general and has been studied for several graph classes. In this chapter it is shown that the problem is NP-complete, even for split graphs, a subclass of P_5-free graphs. In contrast, the main result is that it is linear-time solvable for P_4-free graphs, commonly known as cographs.

This result was initially presented at the 39th International Symposium on Mathematical Foundations of Computer Science (MFCS 2014), Budapest, Hungary [57] and has been published in Discrete Applied Mathematics [58].

2.1 Introduction

We consider *parallel knock-out schemes* for finite undirected graphs with no self-loops and no multiple edges. The *parallel knock-out number* of a graph G, denoted by $\mathrm{pko}(G)$, is the minimum number of rounds in a parallel knock-out scheme that eliminates every vertex of G. If G is not KO-reducible, then $\mathrm{pko}(G) = \infty$. We see an example of a firing round on P_5 in Figure 2.1 in which all but one vertex is eliminated.

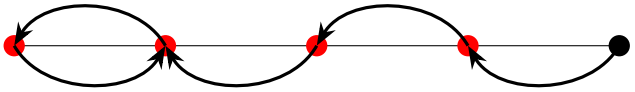


Figure 2.1: An example firing round on P_5.

Examples. The graph P_5, depicted in Figure 2.1 cannot be eliminated. If we denote the vertices $v_1, \ldots, v_5$, we see that v_1 and v_5 must fire upon v_2 and v_4 respectively as they only have one neighbour. v_2 must fire upon v_1 as otherwise v_1 would become an isolated vertex. v_4 must fire upon v_3 for the same reason which leaves nothing to fire at v_5. Some graphs can always be knocked out; every graph G with a Hamiltonian cycle has $\mathrm{pko}(G) = 1$, as each vertex can select its successor on a Hamiltonian cycle C of G after fixing some orientation of C. Also every graph G with a perfect matching has $\mathrm{pko}(G) = 1$, as each vertex can select its matching neighbour in the perfect matching. In fact it is not difficult to see [11] that a graph G has $\mathrm{pko}(G) = 1$ if and only if G contains a *[1,2]-factor*, that is, a spanning subgraph in which every component is either a cycle or an edge. We see an example of this in Figure 2.2.

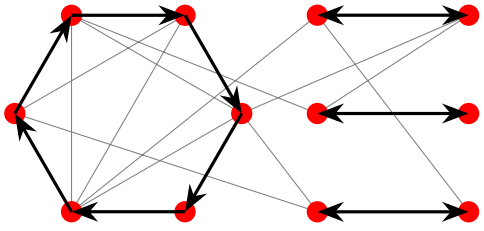


Figure 2.2: An example of a 1 round scheme on a graph with a $[1, 2]$-factor.

We study the computational complexity of the Parallel Knock-Out problem, which is the problem of deciding whether a given graph is KO-reducible.

Parallel Knock-Out	
Instance:	a graph $G = (V, E)$.
Question:	is G KO-reducible?

Our main motivation is the close relationship with cycles and matchings as illustrated by the above examples. We also consider the variant in which the number of rounds permitted is fixed. This problem is known as the k-Parallel Knock-Out problem, which has as input a graph G and ask whether $\text{pko}(G) \leq k$ for some fixed integer k (i.e. that is not part of the input).

k-Parallel Knock-Out	
Instance:	a graph $G = (V, E)$.
Question:	is $\text{pko}(G) \leq k$?

2.1.1 Known Results

The 1-Parallel Knock-Out problem is equivalent [11] to testing whether a graph has a $[1, 2]$-factor, which is well-known to be polynomial-time solvable (see e.g. [13] for a proof). However, both the problems Parallel Knock-Out and k-Parallel Knock-Out with $k \geq 2$ are NP-complete even for bipartite graphs [13]. On the other hand, it is known that Parallel Knock-Out and k-Parallel Knock-Out (for all $k \geq 1$) can be solved in $\mathcal{O}(n^{3.5} \log^2 n)$ time on trees [11]. Broersma et al. in [12] gave an $\mathcal{O}(n^{5.376})$ time algorithm for solving Parallel Knock-Out on n-vertex claw-free graphs ($K_{1,3}$-free graphs). Later this was improved to an $\mathcal{O}(n^2)$ time algorithm for almost claw-free graphs (which generalise the class of claw-free graphs) [59]. The latter paper also gives a full characterisation of connected almost claw-free graphs that are KO-reducible. In particular it shows that every KO-reducible almost claw-free graph has parallel

knock-out number at most 2. In general, KO-reducible graphs (even KO-reducible trees [11]) may have an arbitrarily large parallel knock-out number. Broersma et al. [12] showed that a KO-reducible n-vertex graph G has $\text{pko}(G) \leq \min\{-\frac{1}{2} + (2n - \frac{7}{4})^{\frac{1}{2}}, \frac{1}{2} + (2\alpha - \frac{7}{4})^{\frac{1}{2}}\}$ (where α denotes the size of a largest independent set in G). This bound is asymptotically tight for complete bipartite graphs [11]. Broersma et al. [12] also showed that every KO-reducible graph with no induced $(p+1)$-vertex star $K_{1,p}$ has parallel knock-out number at most $p-1$.

2.1.2 Our Results

To date the only graph classes of unbounded tree-width for which Parallel Knock-Out is known to be polynomial-time solvable are complete bipartite graphs [11] and almost claw-free graphs [59], and we aim to identify further such classes. In particular we want to address the open problem of whether Parallel Knock-Out is polynomial-time solvable on graph classes whose clique-width is bounded by a constant. This seems a very challenging problem, and in this paper we focus on graphs of clique-width at most 2 (which may have arbitrarily large tree-width). It is known that a graph has clique-width at most 2 if and only if it is a cograph [22]. Cographs are also known as P_4-free graphs (a graph is called P_k-free if it has no induced k-vertex path).

In Section 2.3 we give a linear-time algorithm for solving the Parallel Knock-Out problem on cographs. The first step of the algorithm is to compute the cotree of a cograph. The cotree is a decomposition of a cograph where each leaf represents a vertex and every other node represents either the join or disjoint union of its children. The algorithm then traverses the cotree twice. The first time to compute how many "free firings" are available outside a certain subgraph before all vertices must fire internally. The second time to compute to what extent the subgraphs can be reduced by themselves without "firings" from outside. In this way it will be verified whether the whole graph is KO-reducible. In Section 2.4

we prove that both the Parallel Knock-Out problem and the k-Parallel Knock-Out problem ($k \geq 2$) are NP-complete even for split graphs. Because split graphs are P_5-free, our results imply a dichotomy result for the computational complexity of the Parallel Knock-Out problem restricted to P_k-free graphs, as shown in Section 2.5, where we also give some (other) open problems.

2.2 Preliminaries

Let G_1 and G_2 be two disjoint graphs. The *join* operation $\otimes$ adds an edge between every vertex of G_1 and every vertex of G_2. The *union* operation $\oplus$ creates the disjoint union of G_1 and G_2 (note that we may also write $G_1 + G_2$).

It is well known (see, for example, [10]) that a graph G is a cograph if and only if G can be generated from K_1 by a sequence of operations, where each operation is either a join or a union. Such a sequence corresponds to a decomposition tree, which has the following properties:

1. its root r corresponds to the graph $G_r = G$;
2. every leaf x of it corresponds to exactly one vertex of G, and vice versa, implying that x corresponds to a unique single-vertex graph G_x;
3. every internal node x has at least two children, is either labelled $\oplus$ or $\otimes$, and corresponds to an induced subgraph G_x of G defined as follows:
   - if x is a $\oplus$-node, then G_x is the disjoint union of all graphs G_y where y is a child of x;
   - if x is a $\otimes$-node, then G_x is the join of all graphs G_y where y is a child of x.

A cograph G may have more than one such tree but has exactly one unique tree [18], called a *cotree*, if the following additional property is required:

4. Labels of internal nodes on the (unique) path from any leaf to r alternate between $\oplus$ and $\otimes$.

For a node x, we refer to the vertex and edge sets of G_x as V_x and E_x respectively. We denote the cotree of a cograph G by T_G and use the following result of Corneil, Perl and Stewart [19] as a lemma.

Lemma 2.1 ([19])**.** *Let G be a graph with n vertices and m edges. Deciding if G is a cograph and constructing T_G (if it exists) can be done in time $\mathcal{O}(n + m)$.*

Let G be a graph and let $\{H_1, \ldots, H_p\}$ be a set of graphs. We recall that G is $(H_1, \ldots, H_p)$-*free* if G has no induced subgraph isomorphic to a graph in $\{H_1, \ldots, H_p\}$. If $p = 1$ we may write H_1-free instead of (H_1)-free. A P_4-free graph is also called a *cograph*. A graph G is a *split graph* if its vertex set can be partitioned into a clique and an independent set. Split graphs coincide with $(2K_2, C_4, C_5)$-free graphs [38]; since $2K_2$ is an induced subgraph of P_5, it is clear that any graph containing P_5 as an induced subgraph also contains $2K_2$ and is hence not a split graph. As such we see that every split graph is P_5-free.

We need some formal terminology for parallel knock-out schemes. For a graph $G = (V_G, E_G)$, a *KO-selection* is a function $f : V_G \to V_G$ with $f(v) \in N(v)$ for all $v \in V_G$. If $f(v) = u$, we say that vertex v *fires at* vertex u, or that u *is knocked out* by a *firing* of v. If $u \in U$ for some $U \subseteq V_G$ then the firing is said to be *internal* with respect to U if $v \in U$; otherwise it is said to be *external* (with respect to U).

For a KO-selection f, we define the corresponding *KO-successor* of G as the subgraph of G that is induced by the vertices in $V_G \setminus f(V_G)$; if G' is the KO-successor of G we write $G \rightsquigarrow G'$. Note that every graph without isolated vertices has at least one KO-successor. A sequence

$$G \rightsquigarrow G^1 \rightsquigarrow G^2 \rightsquigarrow \cdots \rightsquigarrow G^s,$$

is called a *parallel knock-out scheme* or *KO-scheme*. A KO-scheme in which G^s is the null graph $(\emptyset, \emptyset)$ is called a *KO-reduction scheme*; in that case G is also called *KO-reducible*. A single step in a KO-scheme is called a *(firing) round*. Recall that the parallel knock-out number of G, $\mathrm{pko}(G)$, is the smallest number of rounds of any KO-reduction scheme, and that if G is not KO-reducible then $\mathrm{pko}(G) = \infty$.

We will use the following result of Broersma et al. [11].

Lemma 2.2 ([11])**.** *Let p and q be two integers with $0 < p \leq q$. Then $K_{p,q}$ is KO-reducible if and only if $\mathrm{pko}(K_{p,q}) \leq p$ if and only if $q \leq \frac{1}{2}p\,(p+1)$.*

2.3 Cographs

In this section we present our algorithm, which we call `Cograph-PKO`, for solving Parallel Knock-Out in linear time on cographs. We present a fully worked example on page 29.

Sketch: We start by giving some intuition. Let G be a cograph. We may assume without loss of generality that G is connected, as otherwise we could consider each connected component of G separately.

We first construct the cotree $T_G = (V_{T_G}, E_{T_G})$. Because G is connected, the root r of T_G is a $\otimes$-node. Recall that $G_r = G$ by definition. We observe that if H is a spanning subgraph of G then $\mathrm{pko}(H) \geq \mathrm{pko}(G)$ since any KO-scheme for H also works on G. Consider a partition (X, Y) of the set of children of r such that

$$p = \sum_{x \in X} |G_x| \leq \sum_{y \in Y} |G_y| = q.$$

Note that G has a spanning complete bipartite graph H with partition classes $\bigcup_{x \in X} V_x$ and $\bigcup_{y \in Y} V_y$. Hence, if $q \leq \frac{1}{2}p(p+1)$ then H, and thus G, is KO-reducible by Lemma 2.2. However, such a partition (X, Y) need not exist, but G might still be KO-reducible. In order to find out, we must analyse the cotree of G at lower levels.

The main idea behind our algorithm is as follows. As mentioned above, the graph G_x corresponding to a join node x has at least one spanning complete bipartite subgraph. We will show that it is sufficient to consider only bipartitions, in which one bipartition class corresponds to a single child z of x. We choose z in such a way that if the corresponding complete bipartite subgraph is unbalanced (with respect to the ratio prescribed in Lemma 2.2) then the vertices of G_z correspond to a "large" bipartition class. We will then try to reduce G_z as much as possible by internal firings only. If G_z cannot be reduced to the empty graph,

then external firings are needed. In particular, some of these external firings will be internal firings for supergraphs of G_z. Hence, we first traverse T_G from top to bottom, starting with the root r, to determine the number of external firings for each graph G_z. Afterwards we can then use a bottom-up approach, starting with the leaves of T_G, to determine the number of vertices a graph G_z can be reduced to by internal firings only. If this number is zero for r then G is KO-reducible; otherwise it is not.

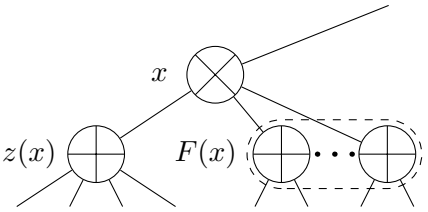


Figure 2.3: A join node within a cotree.

Full Description: Let G be a connected cograph with corresponding cotree $T_G = (V_{T_G}, E_{T_G})$, and let $x \in V_{T_G}$. We say that $|G_x|$ is the *size* of x. We fix a *largest* child of x, that is, a child of x with largest size over all children of x. We denote this child by $z(x)$ (if there is more than one largest child we pick an arbitrary largest one). Let $C(x)$ consist of all other children of x in T_G (so excluding $z(x)$). We write $F(x) = \big(V_{F(x)}, E_{F(x)}\big) = \sum_{y \in C(x)} G_y$. We see an example of this in Figure 2.3.

In our algorithm we recursively define two functions f and l that assign a positive integer to the nodes of V_{T_G}. We write $f(x) = \perp$ or $l(x) = \perp$ if we have not yet assigned an integer $f(x)$ or $l(x)$ to node x; for some nodes x our algorithm might never do this (as we shall see, l will define an integer for a node x if and only if f has defined an integer for x). The meaning of these two functions will be made more clear later. In particular, we will show that $f(x)$ (if defined) is the number of vertices in $V_G \setminus V_x$ adjacent to each vertex of V_x (recall that V_x is the vertex set of G_x). This function will help us in determining how many additional

internal firing rounds we have when we expand G_x to a larger subgraph of G by moving up the tree. The integer $l(x)$ (if defined) is, as we will prove, equal to the smallest number of vertices in G_x that cannot be knocked out internally (that is, within G_x) by any KO-scheme of G. We will show that $l(r)$ is defined, that is, $l(r) \neq \perp$. Hence, there exists a KO-scheme that knocks out all vertices of $V_r = V_G$ if and only if $l(r) = 0$.

`Cograph-PKO`

input : a connected cograph G

output : `yes` if G is KO-reducible; `no` otherwise

Step 1. Compute the size $|G_x|$ for all $x \in V_{T_G}$.

Step 2. Recursively define a function f. Initially set $f(x) := \perp$ for all $x \in V_{T_G}$. Set $f(r) := 0$. Now let x be a vertex in T_G with $f(x) \neq \perp$.

2a. If x is a $\oplus$-node: $f(y) := f(x)$ for all $y \in C(x) \cup \{z(x)\}$.

2b. If x is a $\otimes$-node: $f(z(x)) := f(x) + |F(x)|$ (and keep $f(y) = \perp$ for $y \in C(x)$).

Step 3. Let $B = \{\ell \mid \ell \text{ is a leaf of } T_G \text{ with } f(\ell) \neq \perp\}$.

Step 4. Recursively define a function l. Initially set $l(x) := \perp$ for all $x \in V_{T_G}$. Set $l(\ell) := 1$ for all $\ell \in B$. Now let x be a vertex in T that is either a $\oplus$-node with $l(y) \neq \perp$ for all $y \in C(x) \cup \{z(x)\}$ or a $\otimes$-node with $l(z(x)) \neq \perp$.

4a. If x is a $\oplus$-node: $l(x) := l(z(x)) + \sum_{y \in C(x)} l(y)$.

4b. If x is a $\otimes$-node: $l(x) := \max\{0,\ l(z(x)) - f(x) \cdot |F(x)| - \frac{1}{2}|F(x)|(|F(x)|+1)\}$.

Step 5. If $l(r) = 0$ then return `yes`; otherwise return `no`.

Note that for some $x \in V_{T_G}$, it may happen indeed that $f(x) = \perp$ or $l(x) = \perp$ holds (for example, if x is a leaf node not in B then $l(x) = \perp$).

Example. We illustrate the working of `Cograph-PKO` by applying it to the graph G of Figure 2.4 which leads to Figure 2.5 on page 31, in which the size, f-value and l-value are displayed except for the leaves and all y-nodes except y_3. The other y-nodes have the same f-value and l-value as y_3, namely equal to $\perp$ (our algorithm does not need to compute f and l for these nodes). Because $l(r) = 0$ we conclude that G is KO-reducible. Indeed this can also be seen as follows.

Round 1: $v_1, \ldots, v_6$ fire at v_{17}; v_7 fires at v_1; v_8 fires at v_2; v_9 and v_{13} fire at each other; v_{10} and v_{14} fire at each other; v_{11} and v_{15} fire at each other; v_{12} and v_{16} fire at each other; v_{17} fires at v_3.

Round 2: v_4, v_5, v_6 fire at v_7; v_7 fires at v_4; v_8 fires at v_5.

Round 3: v_6 and v_8 fire at each other.

In order to prove correctness of `Cograph-PKO` we need some new terminology and a number of lemmas. Let x be a node in T_G. Recall that V_x is the vertex set of G_x. We say that a vertex $v \in V_G$ is *complete* to a set $U \subseteq V_G$ with $v \notin U$ if v is adjacent to all vertices of U.

We point out that once $f(x)$ has been given a value, this value is never changed.

Lemma 2.3. *Let $x \in V_{T_G}$ with $f(x) \neq \perp$. The following two statements hold:*

(i) any vertex in $V_G \setminus V_x$ adjacent to a vertex of V_x is complete to V_x;

(ii) the number of vertices in $V_G \setminus V_x$ complete to V_x is equal to $f(x)$.

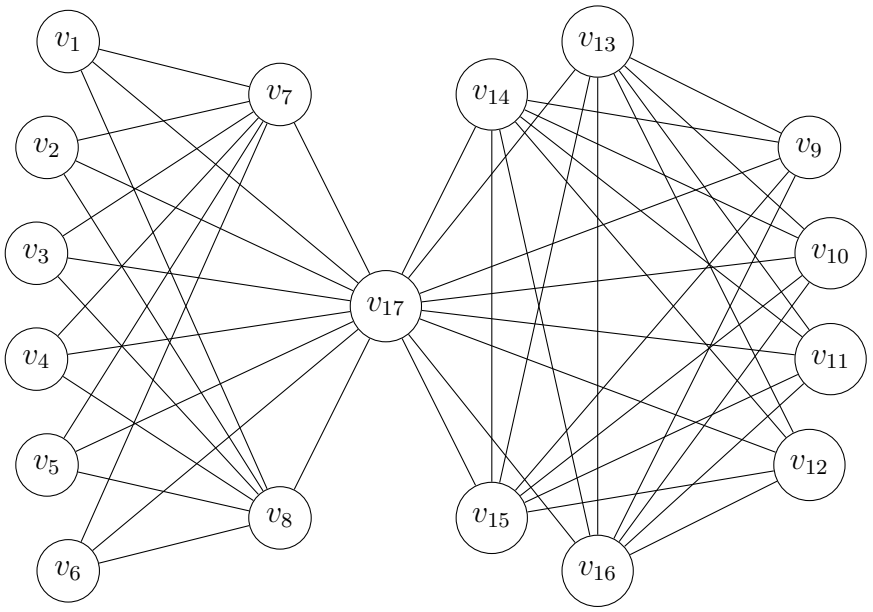


Figure 2.4: An example of a cograph G.

Proof. Let $x \in V_{T_G}$ with $f(x) \neq \bot$. Statement (i) follows from the definition of T_G. We prove (ii) as follows. Let $\text{dist}(x, r)$ denote the distance between x and r (the root node) in T_G. We use induction on $\text{dist}(x, r)$. The claim is true for $\text{dist}(x, r) = 0$ because in that case $x = r$, $V_G \setminus V_x = \emptyset$ and we recall that $f(r) = 0$ by definition.

Let $\text{dist}(x, r) \geq 1$. Then x has a parent in T_G. Denote this parent by x'. By the induction hypothesis, $f(x')$ is equal to the number of vertices not in $G_{v'}$ that are complete to $V_{x'}$. Because V_x is contained in $V_{x'}$, these vertices are complete to V_x as well. Suppose that x is a $\oplus$-node. Then x' is a $\otimes$-node; the fact that $f(x) \neq \bot$ means that by construction, $x = z(x')$. This means that all vertices in $F(x')$ are complete to V_x. Hence, the total number of vertices in $V_G \setminus V_x$ that are complete to V_x is equal to $f(x') + |F(x')| = f(x)$. Suppose that x is a $\otimes$-node. Then x' is a $\oplus$-node. This means that no vertex in $F(x')$ is adjacent to a vertex in V_x. Hence, the total number of vertices in $V_G \setminus V_x$ that are complete to V_x is equal to $F(x') = f(x)$. □

The following lemma follows directly from the construction of our algorithm on page 28.

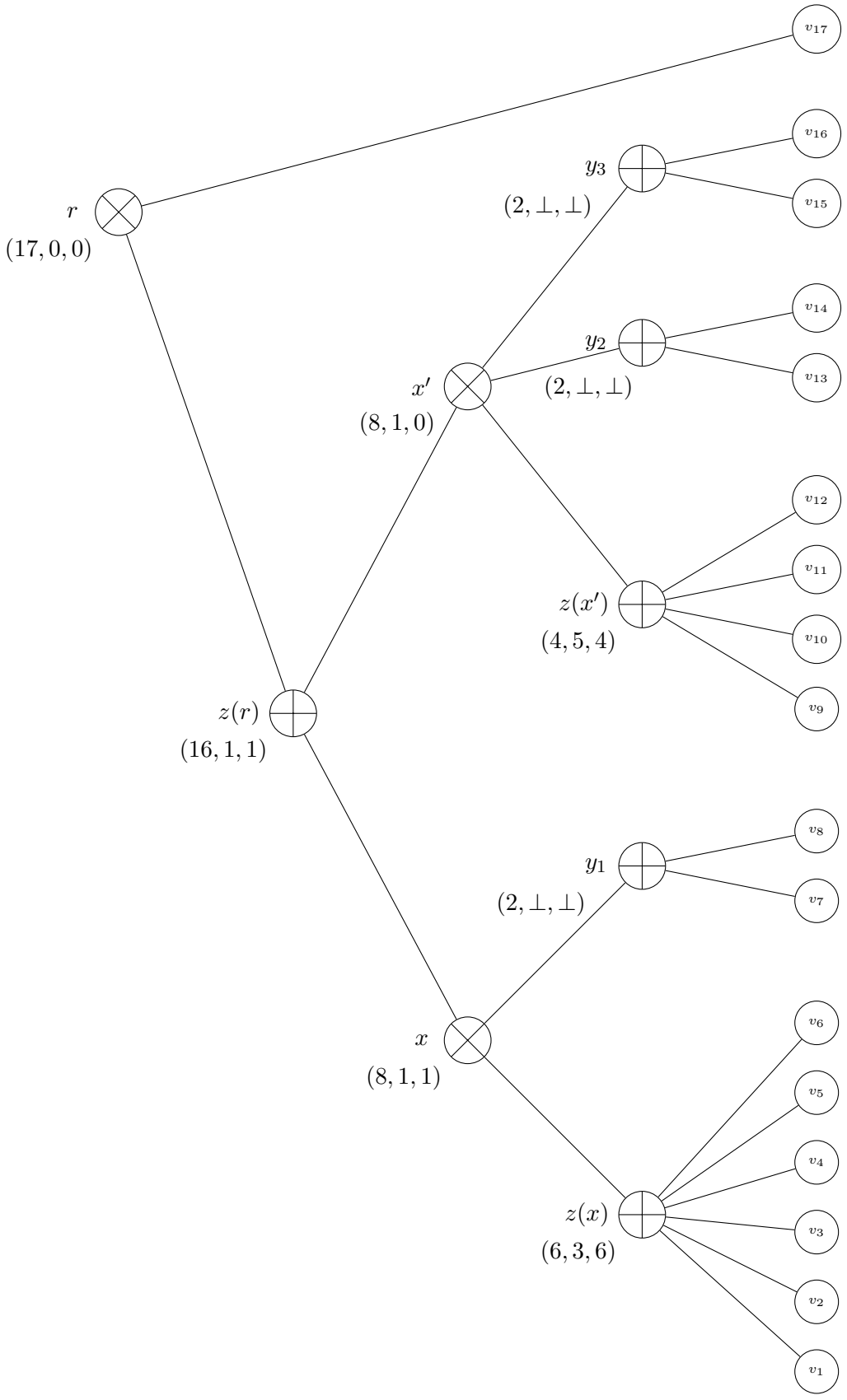


Figure 2.5: The cotree T_G of the graph G of Figure 2.4. For each node x we show the triple $(|V_x|, f(x), l(x))$ with f-values and l-values obtained from executing `Cograph-PKO`. Note that $B = \{v_1, v_2, v_3, v_4, v_5, v_6\} \cup \{v_9, v_{10}, v_{11}, v_{12}\}$.

Lemma 2.4. *Let $x \in V_{T_G}$. Then $l(x) \neq \perp$ if and only if $V_x \cap B \neq \emptyset$.*

Before we continue, we first introduce a number of new notions, namely of x-pseudo-KO-selection, x-pseudo-KO-successor, x-pseudo-KO-scheme, x-pseudo-reducibility and pseudo(x). Each of these notions play an important role in proving the correctness of our algorithm.

Let x be a node in T_G. An *x-pseudo-KO-selection* of G is a function $f : V_x \to V_G$ with $f(v) \in N(v)$ for all $v \in V_G$. We copy some terminology. If $f(v) = u$, we say that v fires at u, or that u is knocked-out by a firing of v. Note that every KO-selection of G is an x-pseudo-KO-selection of G. However, the reverse implication is not true if $x \neq r$, because we do not let any vertices in $V_G \setminus V_x$ fire according to this definition.

For an x-pseudo-KO-selection f, we define the *x-pseudo-KO-successor* of G_x as the subgraph of G induced by $V_G \setminus f(V_x)$. We write $G \rightsquigarrow^x G'$ to denote that G' is an x-pseudo-KO-successor of G. We call a sequence

$$G \rightsquigarrow^x G^1 \rightsquigarrow^x \cdots \rightsquigarrow^x G^s$$

an *x-pseudo-KO-scheme* (where each single step is called a round) of G if in addition there is no vertex of V_x that fires at a vertex of V_x in some round i and at a vertex in $V_G \setminus V_x$ in some round $j > i$. Note that an r-pseudo-KO-scheme of G is a KO-reduction scheme of G. Let G^s_x be the subgraph of G^s induced by V_x. Then we say that G_x is *x-pseudo-reducible* to G^s_x. Define pseudo(x) as the number of vertices in a smallest graph to which G_x is x-pseudo-reducible and say that a corresponding x-pseudo-KO-scheme of G is *optimal*.

Lemma 2.5. *The cograph G is KO-reducible if and only if* pseudo$(r) = 0$.

Proof. Recall that $V_r = V_G$. Then the statement of the lemma holds because every KO-reduction scheme of G (if there exists one) is an r-pseudo-KO-scheme with pseudo$(r) = 0$, and vice versa. □

The following lemma is crucial for the correctness of our algorithm.

Lemma 2.6. *Let $x \in V_{T_G}$ be a $\otimes$-node with $l(x) \neq \perp$. Then $l(x) = \text{pseudo}(x)$.*

Proof. Let $x \in V_{T_G}$ be a $\otimes$-node with $l(x) \neq \perp$. By Lemma 2.4, $V_x \cap B \neq \emptyset$. We write $z = z(x)$. Let $|V_z| = q$ and $|F(x)| = p$. This enables us to write:

$$\begin{aligned} l(x) &= \max\{0, l(z) - f(x) \cdot |F(x)| - \tfrac{1}{2}|F(x)|(|F(x)|+1)\} \\ &= \max\{0, l(z) - f(x) \cdot p - \tfrac{1}{2}p(p+1)\}. \end{aligned}$$

Note that $q \geq 1$ and $p \geq 1$ by the definition of a $\otimes$-node.

Let d denote the number of internal $\otimes$-nodes on the longest path from x to a leaf in the subtree of T_G rooted at x. We prove the lemma by induction on d.

I. The Base Case.

Let $d = 0$. Then every child of x is either a leaf itself or all its children are leaves.

First suppose z is a leaf. Because $V_x \cap B \neq \emptyset$, we find that $z \in B$. Hence, $l(z) = 1$. Then, as $p \geq 1$, we find that $l(z) - f(x) \cdot p - \frac{1}{2}p(p+1) \leq 0$. Hence, $l(x) = 0$. Note that $q = 1$. Because z is a largest child of x, all children of x are leaves. Hence, G_x is a complete graph on $p+1$ vertices. This means that G_x is KO-reducible. We conclude that $\text{pseudo}(x) = 0 = l(x)$.

Now suppose that z is not a leaf. Then z has at least two children (which are all leaves since $d = 0$). Hence, $q \geq 2$. Because $V_x \cap B \neq \emptyset$, every child of z is in B, that is, $V_z = B$ is an independent set, in particular, $q = |B|$. Because $l(\ell) = 1$ for every $\ell \in B$, this means that $l(z) = |B| = q$. We distinguish three cases.

Case 1. $q < p$.

Then $l(z) - f(x) \cdot p - \frac{1}{2}p(p+1) = q - f(x) \cdot p - \frac{1}{2}p(p+1) \leq 0$. Hence, $l(x) = 0$.

Let $y_1, \ldots, y_r, z$ be the children of x for some $r \geq 1$. In fact, because $2 \leq q < p$ and z is the largest child of x, we find that $r \geq 2$. Assume that $|V_{y_1}| \geq \cdots \geq |V_{y_r}|$.

By definition, $q = |V_z| \geq |V_{y_1}|$. Because $q < p$, we can pick a set D of $q - |V_{y_1}| \geq 0$ vertices of $V_{F(x)} \setminus V_{y_1}$. We define $T_1 = V_{y_1} \cup D$ and $T_i = V_{y_i} \setminus D$ for $i = 2, \ldots, r$. Note that $|T_1| = q$.

Let $\{|T_1|, \ldots, |T_r|\} = \{j_1, \ldots, j_r\}$, without loss of generality we may assume $j_1 \geq \cdots \geq j_r$. Choose s such that $j_s > 0$ and either $s = r$ or $j_{s+1} = 0$. Because $|T_1| = q$, we find that $j_1 = q$. We recall that for any $i > j \geq 2$, every vertex of T_i is adjacent to every vertex of T_j and is adjacent to every vertex of V_{y_1}. We partition V_x into s subsets. For the first subset we pick j_s vertices from V_z, j_s vertices from $V_{y_1} \subseteq T_1$ and also j_s vertices from each non-empty T_i with $i \geq 2$. Since $q < p$, at least one other set T_i, besides T_1, is nonempty. As such we see that the graph induced by the union of all these vertices has a Hamiltonian cycle as seen in Figure 2.6. We remove all chosen vertices. Then, for the second subset of our partition, we pick $j_{s-1} - j_s$ vertices from V_z and also $j_{s-1} - j_s$ vertices from V_{y_1} and each T_i $(i > 2)$ that is not yet empty. The graph induced by the union of all chosen vertices has a Hamiltonian cycle while there exists a non-empty set T_i $(i \geq 2)$. We repeat this procedure until all sets T_i $(i \geq 2)$ are empty, at which point we see that we have $q - j_2$ vertices remaining in both T_1 and V_z which can be eliminated via a perfect matching. In this way we have found a $[1, 2]$-factor of G_x. Consequently, $\text{pko}(G_x) = 1$. Hence, $\text{pseudo}(x) = 0 = l(x)$.

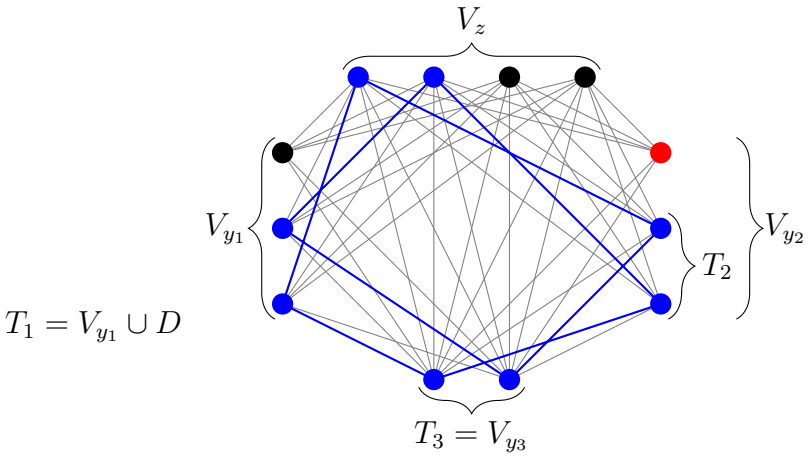


Figure 2.6: An example of G_x where $s = 3$ and $j_s = 2$. The vertex in D is shown in red. We see the vertices chosen for the first subset and a Hamiltonian path through these vertices shown in blue.

Case 2. $q \geq p$ and $l(x) = 0$.

As $l(z) = q$, the assumption that $l(x) = 0$ implies that $q - f(x) \cdot p \leq \frac{1}{2}p(p+1)$. By Lemma 2.3, all vertices in $V_G \setminus V_x$ that are adjacent to V_x are complete to V_x and moreover, the number of such vertices is equal to $f(x)$. This enables us to define the following x-pseudo-KO-scheme. Let all vertices of $F(x)$ fire at different vertices in V_z for the first $f(x)$ rounds. Let all vertices in V_z fire at the same vertex of $V_G \setminus V_x$ for the first $f(x)$ rounds. Note that q decreases in this way. However, we may not need to perform all these rounds: after each round we check whether $p \leq q \leq \frac{1}{2}p(p+1)$. Because $q - f(x) \cdot p \leq \frac{1}{2}p(p+1)$, it will eventually happen that $q \leq \frac{1}{2}p(p+1)$. If it turns out that $q < p$, we slightly adjust the previous round by letting a sufficient number of vertices of $F(x)$ fire at the same vertex in V_z instead of at different vertices, in order to get $p \leq q \leq \frac{1}{2}p(p+1)$. We then apply Lemma 2.2 to knock out the remaining vertices of V_x in at most p additional rounds. Hence $\text{pseudo}(x) = 0 = l(x)$.

Case 3. $q \geq p$ and $l(x) > 0$.

As $l(z) = q$, the assumption that $l(x) > 0$ implies that $q > f(x) \cdot p + \frac{1}{2}p(p+1)$. Recall that, by Lemma 2.3, all vertices in $V_G \setminus V_x$ that are adjacent to V_x are complete to V_x, and moreover, the number of such vertices is equal to $f(x)$. This enables us to define the following x-pseudo-KO-scheme. Let all vertices of $F(x)$ fire at different vertices in V_z for the first $f(x)$ rounds. Let all vertices in V_z fire at the same vertex of $V_G \setminus V_x$ for the first $f(x)$ rounds. Afterwards we can reduce the number of vertices of V_x by at most $\frac{1}{2}p(p+1)$ by letting all vertices of $F(x)$ fire at different vertices in V_z, whereas all vertices in V_z fire at the same vertex of $F(x)$ until $F(x) = \emptyset$. Because $F(x) = \emptyset$ in the end, and in each round we have reduced the maximum number of vertices of the independent set V_z, we find that $\text{pseudo}(x) = q - f(x) \cdot p - \frac{1}{2}p(p+1) = l(z) - f(x) \cdot p - \frac{1}{2}p(p+1) = l(x)$.

II. The Inductive Step

Let $d \geq 1$. Then z is not a leaf as otherwise all children of x are leaves, which

contradicts $d \geq 1$. Consequently, z is a $\oplus$-node. We distinguish two cases.

Case 1. $q < p$.

Observe that $l(z) \leq q$. Then $l(z) - f(x) \cdot p - \frac{1}{2}p(p+1) \leq q - f(x) \cdot p - \frac{1}{2}p(p+1) \leq 0$. Hence, $l(x) = 0$. We repeat the same arguments as for the corresponding case for $d = 0$ to obtain that $\text{pseudo}(x) = 0 = l(x)$. So Case 1 is proven.

Before we consider Case 2, we first analyse the subtree of T_G rooted at x. Let $s_1, \ldots, s_p$ be the children of z with $l(s_i) > 0$ for $i = 1, \ldots, p$ (if such children exist) and let $t_1, \ldots, t_q$ be the children of z with $l(t_i) = 0$ for $i = 1, \ldots, q$ (if such children exist). Note that all children of z are either leaves or $\otimes$-nodes. Let z' be a child of z. If z' is a leaf, then $\text{pseudo}(z') = 1 = l(z')$. If z' is a $\otimes$-node, we may apply the induction hypothesis to find that $\text{pseudo}(z') = l(z')$. In other words, $\text{pseudo}(s_i) = l(s_i)$ for $i = 1, \ldots, p$ and $\text{pseudo}(t_i) = l(t_i)$ for $i = 1, \ldots, q$. Then, because $G_z = G_{s_1} + \cdots + G_{s_p} + G_{t_1} + \cdots + G_{t_q}$, we find that an optimal z-pseudo-KO-scheme mimics the optimal s_i-pseudo-KO-schemes and optimal t_j-pseudo-KO-schemes (we may assume without loss of generality that all external firings outside G_z in a round are always at a single vertex). Hence, $\text{pseudo}(z) = l(s_1) + \cdots + l(s_p) + l(t_1) + \cdots + l(t_q) = l(z)$.

Case 2. $q \geq p$.

We define the following x-pseudo-KO-selection scheme. The firing rounds for the vertices in G_z are according to an optimal z-pseudo-KO-scheme under the following conditions. For the first $f(x)$ rounds any external firings outside G_z are at a single vertex, which is not in G_x. Note that this is possible by Lemma 2.3. Afterwards any external firing outside G_z must be in $F(x)$ and also for such firings we require that they are at a single vertex in every round. The vertices in $F(x)$ fire in each round at different vertices of G_x that are in $G_{s_1} + \cdots + G_{s_p}$ and that are not being fired at by vertices in G_x. They stop firing in a graph G_{s_i} as soon as they have knocked out $l(s_i)$ of its vertices. Note that we are guaranteed a budget of exactly $f(x) \cdot p + \frac{1}{2}p(p+1)$ firings from vertices outside G_z into G_z.

First suppose that $l(z) - f(x) \cdot p \leq \frac{1}{2}p(p+1)$, so $l(x) = 0$. Then we can knock out all $l(z)$ vertices of G_z that cannot be knocked out by internal firings inside G_z. As we still need to knock out the vertices of $F(x)$, we check after each round whether q has decreased such that $p \leq q \leq \frac{1}{2}p(p+1)$ holds. Because $q - f(x) \cdot p \leq \frac{1}{2}p(p+1)$, it will eventually happen that $q \leq \frac{1}{2}p(p+1)$. If it turns out that $q < p$, we slightly adjust the previous round as we did in Case 2 for $d = 0$, in order to get $p \leq q \leq \frac{1}{2}p(p+1)$. We then apply Lemma 2.2 to knock out the remaining vertices of V_x in at most p additional rounds. We conclude that $\text{pseudo}(x) = 0 = l(x)$.

Now suppose that $l(z) - f(x) \cdot p > \frac{1}{2}p(p+1)$, so $l(x) > 0$. Then, by the definition of our x-pseudo-KO-reduction scheme, all vertices in $F(x)$ have fired at different vertices in every round for $f(x) \cdot p + \frac{1}{2}p(p+1)$ rounds. Moreover, all vertices in $F(x)$ are knocked out afterwards. Because $\text{pseudo}(z) = l(z)$ and because we mimicked an optimal z-pseudo-KO-scheme as regards the firings of the vertices of G_z in each round, we cannot improve upon this. As such we conclude that $\text{pseudo}(x) = l(z) - f(x) \cdot p - \frac{1}{2}p(p+1) = l(x)$.

III. Conclusion.

We have proven the base case and the inductive step. Hence we conclude that $\text{pseudo}(x) = l(x)$ for any value of d. This completes the proof of Lemma 2.6. □

Theorem 2.1. *The* Parallel Knock-Out *problem can be solved in $\mathcal{O}(n+m)$ time on cographs with n vertices and m edges.*

Proof. Let G be a cograph with n vertices and m edges. If G is disconnected we consider each connected component of G separately. Hence, assume that G is connected.

We construct T_G. Run `Cograph-PKO` with input G. By Lemma 2.4, we find that $l(r) \neq \perp$. Hence, we may apply Lemma 2.6 to find that $l(r) = \text{pseudo}(r)$. By Lemma 2.5, we find that G is KO-reducible if and only if $\text{pseudo}(r) = 0$. As `Cograph-PKO` outputs a yes-answer if and only if $l(r) = 0$, we find it is correct.

It remains to show that `Cograph-PKO` runs in linear time. We can perform Step 1 in a bottom-up approach starting from the leaves of T_G. So, Steps 1-3 each visit each node at most once. This means that every node of x is visited at most three times in total. Because every co-tree has at most $n + n - 1 = 2n - 1$ vertices, we find that the running time of `Cograph-PKO` is $\mathcal{O}(n)$. Because constructing T_G costs time $\mathcal{O}(n + m)$ by Lemma 2.1, the total running time is $\mathcal{O}(n + m)$. □

2.4 Split Graphs

We show the following result, the proof of which is (partially) based on the NP-hardness proof of 2-Parallel Knock-Out for bipartite graphs from [13].

Theorem 2.2. *The* Parallel Knock-Out *problem and, for any $k \geq 2$, the* k-Parallel Knock-Out *problem are* NP*-complete for split graphs.*

Proof. First consider the Parallel Knock-Out problem. We reduce from the Dominating Set problem, which is well known to be NP-complete (see [39]). This problem takes as input a graph $G = (V, E)$ and a positive integer p. We may assume without loss of generality that $p \leq |V|$. The question is whether G has a dominating set of cardinality at most p.

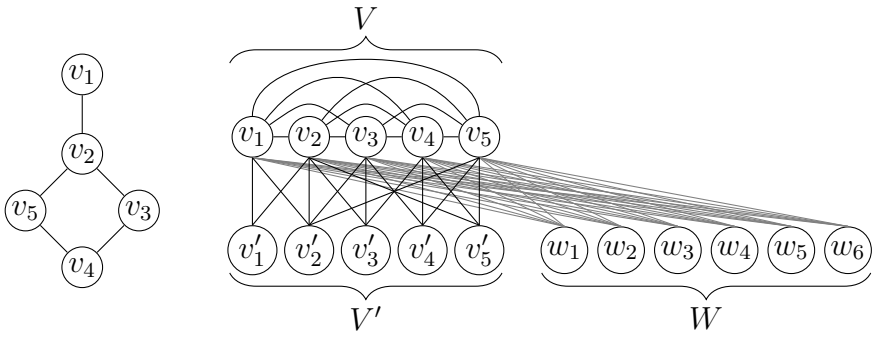


Figure 2.7: An example of a graph G (left) and the split graph G' constructed from an instance $(G, 2)$ of Dominating Set (right). (Note that the set V is a clique.)

From an instance (G, p) of Dominating Set we construct a split graph $G' = (V_{G'}, E_{G'})$ as follows. Let $V_G = \{v_1, \ldots, v_n\}$. We let $V_{G'}$ consist of three mutually disjoint sets: the set $V = \{v_1, \ldots, v_n\}$, a set $V' = \{v'_1, \ldots, v'_n\}$ and a set $W = \{w_1, \ldots, w_r\}$ where $r = \frac{1}{2}(n-p)(n-p+1)$. We define $E_{G'}$ as follows. First we add the edges $v_i v'_i$ for $i = 1, \ldots, n$. For all $i \neq j$, we add the edges $v_i v'_j$ and $v_j v'_i$ if and only if $v_i v_j$ is an edge in E_G. We also add an edge between every v_i and every w_j. Finally, we add an edge between any two vertices in V. For an example see Figure 2.7. Observe that G' is indeed a split graph in which V is a clique of

size n and $V' \cup W$ is an independent set of size $n + r$. We claim that G has a dominating set of size at most p if and only if G' is KO-reducible.

First suppose G has a dominating set D of size at most p. Because $p \leq |V|$, we may assume without loss of generality that $D = \{v_1, \ldots, v_p\}$. We construct a KO-reduction scheme of G' as follows. In the first round let every vertex $v_i \in V$ fire at $v'_i \in V'$. For $i = 1, \ldots, p$, let v'_i fire at v_i. For $i = p + 1, \ldots, n$ let v'_i fire at an arbitrary vertex in D, which is possible because D is a dominating set of G. Finally, let every vertex in W fire at an arbitrary vertex in D as well; this is possible by the construction of G'. The resulting (split) graph G'' consists of a clique $V \setminus D$ of size $n - p$ and the independent set W of size $\frac{1}{2}(n - p)(n - p)$. Because there is an edge between every vertex in V and every vertex in W, we find that G'' is KO-reducible by Lemma 2.2.

Now suppose G' is KO-reducible. Consider a KO-reduction scheme of G'. Let D be the subset of vertices of G' that belong to V and that are knocked out in the first round. Because each vertex must fire at a neighbour, and vertices of V' can only fire at vertices of V (as they have neighbours only in V), we deduce that the vertices of D form a dominating set of G.

In order to complete the proof it remains to prove that $|D| \leq p$. For contradiction, suppose that $|D| \geq p + 1$. Let $V_1 = V \setminus D$ be the subset of V consisting of vertices not knocked out in the first round. Because $|D| \geq p + 1$, we obtain $|V_1| = |V| - |D| \leq n - p - 1$. Let V^* and W^* be the subsets of V' and W, respectively, that consist of vertices not knocked out in the first round. Vertices in $V' \cup W$ can only be knocked out by vertices of V. Moreover, the total number of vertices that V can knock out in the first round is at most $|V| = n$. This means that $V^* \cup W^*$ is an independent set of size

$$|V^* \cup W^*| = |V^*| + |W^*| \geq |V'| + |W| - n = \tfrac{1}{2}(n - p)(n - p + 1).$$

However, as in every round the size of V_1 is reduced by at least 1, the maximum number of vertices in $V^* \cup W^*$ that V_1 can knock out is at most

$$(n-p-1)+(n-p-2)+\cdots+1 < \frac{1}{2}(n-p)(n-p+1).$$

Hence, the scheme is not a KO-reduction scheme of G'. This is a contradiction, and we have completed the proof for Parallel Knock-Out.

Now let $k \geq 2$ and consider the k-Parallel Knock-Out problem. We use the same reduction and the same arguments as for Parallel Knock-Out after changing the size of W into $r := (n-p)+(n-p-1)+\cdots+(n-p-k+2)$. □

2.5 Conclusions

We have shown in Theorem 2.1 that Parallel Knock-Out is linear-time solvable for P_4-free graphs. We have also shown in Theorem 2.2 that Parallel Knock-Out and, for any $k \geq 2$, k-Parallel Knock-Out are NP-complete for split graphs. Because split graphs are $(2K_2, C_4, C_5)$-free [38], they are P_5-free. Hence, Theorems 2.1 and 2.2 have the following consequence.

Corollary 2.1. *The* Parallel Knock-Out *problem restricted to P_r-free graphs is linear-time solvable if $r \leq 4$ and* NP*-complete if $r \geq 5$.*

Whether it is possible to compute $\mathrm{pko}(G)$ in polynomial time for cographs is still an open problem. It is natural to ask whether this can be solved by adjusting our algorithm `Cograph-PKO` to solve k-Parallel Knock-Out on cographs in polynomial time for every fixed integer k. We investigated this approach, but encountered the following problem. With unlimited rounds, either all the vertices in $F(x)$ are used up or $G_{z(x)}$ can be reduced entirely so an adjustment can be made such that $F(x)$ and $G_{z(x)}$ knock each other out in the final round. With a restriction on the number of rounds, the assumption that $F(x)$ will always be entirely eliminated is no longer valid, since there may be survivors in both $F(x)$ and $G_{z(x)}$ and determining their optimal firing is not trivial.

Also recall that cographs are exactly those graphs that have clique-width at most 2 [22]. Can we solve Parallel Knock-Out in polynomial time for graphs of clique-width at most 3? For this we could start by considering the class of distance-hereditary graphs, which have clique-width at most 3 [47]. Distance-hereditary graphs are completely decomposable with respect to a so-called split decomposition [49], a graph decomposition introduced by Cunningham and Edmonds [23] which may be useful for our purposes.

We also do not know whether there is a constant c such that Parallel Knock-Out is NP-complete for graphs of clique-width at most c. However, it is known

that the related NP-complete problem Hamiltonian Cycle, which tests whether a graph has a Hamiltonian cycle, is polynomial-time solvable on any graph class whose clique-width is bounded by a constant (this follows from combining results of [56, 98], also see [29]).

A different direction from above for extending our results would be to classify the complexity of Parallel Knock-Out restricted to H-free graphs. The complexity status is open even for small graphs $H \in \{4P_1, 2P_1+2P_2, P_1+P_3, K_{1,4}\}$.

Chapter 3

Minimal Disconnected Cut on Planar Graphs

On a connected graph $G = (V, E)$, a subset $U \subseteq V$ is called a *disconnected* cut if U disconnects the graph and the subgraph induced by U is disconnected as well. On a connected graph $G = (V, E)$, a subset $U \subseteq V$ is called a *minimal disconnected* cut if U is a disconnected cut and $G[(V \backslash U) \cup \{u\}]$ is connected for every $u \in U$. Ito et al. [54] showed that the problem of finding a minimal disconnected cut in a graph is NP-hard in general but its computational complexity is unknown for planar graphs.

In this chapter, we show that the problem of finding a minimal disconnected cut is polynomial-time solvable on 3-connected planar graphs but NP-hard for 2-connected planar graphs. Our technique for the first result makes use of a structure characterisation which we show is apparent in all 3-connected $K_{3,3}$-free-minor graphs if and only if it permits a minimal disconnected cut.

In addition we show that the problem of finding a minimal connected cut of size at least 3 is NP-hard for 2-connected apex graphs, and hence for 2-connected planar graphs as well. Finally, we relax the notion of minimality and prove that the problem of finding a semi-minimal disconnected cut is still polynomial-time solvable on planar graphs.

This result was initially presented at the 20th International Symposium on Fundamentals of Computation Theory (FCT 2015), Gdańsk, Poland [60] and has been published in Networks [61].

3.1 Introduction

A *cutset* or *cut* in a connected graph is a subset of its vertices whose removal disconnects the graph. The problem Stable Cut is that of testing whether a connected graph has a cut that is an independent set. Le, Mosca, and Müller [68] proved that this problem is NP-complete even for K_4-free planar graphs with maximum degree 5. A connected graph $G = (V, E)$ is *k-connected* for some integer k if $|V| \geq k+1$ and every cut of G has size at least k. It is not hard to see that if one can solve Stable Cut for 3-connected planar graphs in polynomial time then one can do so for all planar graphs (in particular the problem is trivial if the graph has a cut-vertex or a cut set of two vertices that are non-adjacent). Hence, the problem is NP-complete for 3-connected planar graphs.

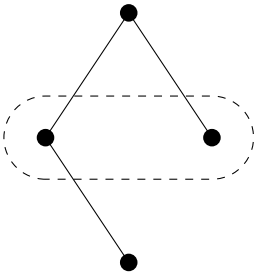


Figure 3.1: The graph P_4 with a disconnected cutset.

Due to the above it is a natural question whether one can relax the condition on the cut to be an independent set. This leads to the following notion. For a connected graph $G = (V, E)$, a subset $U \subseteq V$ is called a *disconnected* cut if U disconnects the graph and the subgraph induced by U is disconnected as well, that is, has at least two (connected) components. This problem is NP-compete in general [76] but polynomial-time solvable on planar graphs [55]. However, the property of the cut being disconnected can be viewed to be somewhat artificial if one considers the 4-vertex path $P_4 = p_1p_2p_3p_4$, which has two disconnected cuts, namely $\{p_1, p_3\}$ and $\{p_2, p_4\}$. We see an example of this in Figure 3.1. Both these cuts contain a vertex, namely p_1 and p_4, respectively, such that putting this vertex out of the cut and back into the graph keeps the graph disconnected.

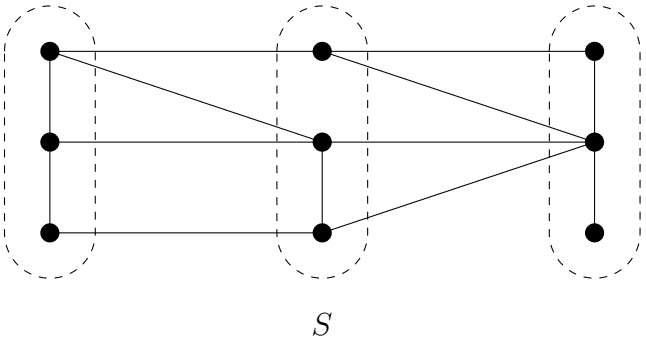


Figure 3.2: An example of a planar graph with a minimal disconnected cut, namely the set S.

Therefore, Ito et al. [54] defined the notion of a *minimal disconnected* cut of a connected graph $G = (V, E)$, that is, a disconnected cut U so that $G[(V \setminus U) \cup \{u\}]$ is connected for every $u \in U$ (more generally, we call a cut that satisfies the later condition a *minimal* cut). We note that every vertex of a minimal cut U of a connected graph $G = (V, E)$ is adjacent to every component of $G[V \setminus U]$. See Figure 3.2 for an example of a planar graph with a minimal disconnected cut. The corresponding decision problem is defined as follows.

Minimal Disconnected Cut	
Instance:	a connected graph $G = (V, E)$.
Question:	does G have a minimal disconnected cut?

Ito et al. [54] showed that Minimal Disconnected Cut is NP-complete. However its computational complexity remained open for planar graphs. As a graph has a stable cut if and only if a graph has a minimal stable cut, the problem of deciding whether a graph has a minimal stable cut is NP-complete for any graph class (and thus for the class of planar graphs) for which Stable Cut is NP-complete. In contrast, the problem of deciding whether a graph has a minimal cut (that may be connected or disconnected) is polynomial-time solvable: for any graph, given a vertex cut U we can remove vertices from U one by one until the remaining vertices in U form a minimal cut.

3.1.1 Our Results

As a start we observe that Minimal Disconnected Cut is polynomial-time solvable for outerplanar graphs (as these graphs do not contain $K_{2,3}$ as a minor, any minimal cut has size at most 2). In Section 3.3 we prove that Minimal Disconnected Cut is also polynomial-time solvable on 3-connected planar graphs. The technique used by Ito et al. [55] for solving Disconnected Cut in polynomial time for planar graphs was based on the fact that a planar graph either has its treewidth bounded by some constant or else contains a large grid as a minor. However, grids (which are 3-connected planar graphs) do not have minimal disconnected cuts. Hence, we need to use a different approach, which we describe below.

We first provide a structural characterisation of minimal disconnected cuts for the class of 3-connected $K_{3,3}$-minor-free graphs, which contains the class of 3-connected planar graphs. In particular we show that any minimal disconnected cut of a 3-connected planar graph G has exactly two components and that these components are paths. A graph G contains a graph H as a contraction if H can be reach from G by a sequence of edge contractions. In order to find such a cut we prove that it suffices to test whether G contains, for some fixed integer r, the biclique $K_{2,r}$ as a contraction. We show that G has such a contraction if and only if its dual contains the multigraph D_r, which is obtained from the r-vertex cycle by replacing each edge by two parallel edges, as a subdivision (which we define in Section 3.2; see Figure 3.4 on page 57 for examples of D_r). We then present a characterisation of any graph that contains such a subdivision. Next we use this characterisation to prove that the corresponding decision problem of finding a multigraph D_r as a subdivision for some $r \geq 2$ is polynomial-time solvable even on general graphs.

In Section 3.4 we give our second result, namely that, contrary to Disconnected

Cut, which is polynomial-time solvable for planar graphs [55], Minimal Disconnected Cut stays NP-complete for the class of 2-connected planar graphs. Our proof is based on a reduction from Stable Cut and as such is different from the NP-hardness proof for general graphs [54], the gadget of which contains large cliques.

In Section 3.4 we also show that the problem of finding a minimal *connected* cut of size at least 3 is NP-complete for 2-connected *apex graphs* (graphs that can be made planar by deleting one vertex); to the best of our knowledge the computational complexity of this problem has not yet been determined even for general graphs. We note that the problem of finding whether a graph contains a (not necessarily minimal) connected cut of size at most k that separates two given vertices s and t is linear-time FPT when parameterised by k [77].

In Section 3.5 we consider a generalisation of (minimal) disconnected cuts and stable cuts. For a family of graphs $\mathcal{H}$, a connected graph has a *(minimal) $\mathcal{H}$-cut* if it has a (minimal) cut that induces a graph in $\mathcal{H}$. This leads to the corresponding decision problems $\mathcal{H}$-Cut and Minimal $\mathcal{H}$-Cut. For instance, we can describe (Minimal) Stable Cut as (Minimal) $\{P_1, 2P_1, 3P_1, \ldots\}$-Cut. Moreover, if $\mathcal{H}$ consists of all disconnected graphs, we obtain the (Minimal) Disconnected Cut problem. The problem of finding minimum (that is, smallest) $\mathcal{H}$-cuts that separate two given vertices s and t has been studied from a parameterised point of view for various graph families $\mathcal{H}$ by Heggernes et al. [51]. We show some initial results for Minimal $\mathcal{H}$-Cut, which provide some further insights in our main results.

In Section 3.6 we relax the notion of minimality for cut sets as follows. If a cut U of a graph $G = (V, E)$ is minimal, each of its vertices is adjacent to every component in $G[V \setminus U]$. What if instead we demand that each vertex $u \in U$ is adjacent to at least two (but maybe not all) components of $G[V \setminus U]$? This leads to the following definition. A disconnected cut U of a connected graph

$G = (V, E)$ is *semi-minimal* if $G[(V \setminus U) \cup \{u\}]$ contains fewer components than $G[V \setminus U]$ for every $u \in U$. The corresponding decision problem, which is known to be NP-complete [54], is called Semi-Minimal Disconnected Cut. Note that there exist graphs with a disconnected cut, such as the P_4, that have no semi-minimal disconnected cut. Because for planar graphs Minimal Disconnected Cut is NP-complete and Disconnected Cut is polynomial-time solvable, it is a natural question to determine the complexity of Semi-Minimal Disconnected Cut for planar graphs. We adapt the proof for Disconnected Cut to show that Semi-Minimal Disconnected Cut is also polynomial-time solvable on planar graphs.

We finish this chapter with some further observations and open problems in Section 3.7.

3.1.2 Related Work

Vertex cuts play an important role in graph connectivity. In the literature various kinds of vertex cuts, besides stable cuts, have been studied extensively and we briefly survey a number of results below that have not been mentioned yet.

A cut U of a graph $G = (V, E)$ is a clique cut if $G[U]$ is a clique, a k-clique cut if $G[U]$ has a spanning subgraph consisting of k complete graphs; a strict k-clique cut if $G[U]$ consists of k components that are complete graphs; and a matching cut if $E_{G[U]}$ is a matching. We note that a clique cut is equivalent to a 1-clique cut and to a strict 1-clique cut. It follows from a classical result of Tarjan [92] that determining whether a graph has a clique cut is polynomial-time solvable. Whitesides [99] and Cameron et al. [14] proved that the problem of testing whether a graph has a k-clique cut is solvable in polynomial time for $k = 1$ and $k = 2$, respectively. Cameron et al. [14] also proved that testing whether a graph has a strict 2-clique cut can be solved in polynomial time. As mentioned the problem of testing whether a graph has a stable cut is NP-complete. This was first shown

for general graphs by Chvátal [15]. Also the problem of testing whether a graph has a matching cut is NP-complete. This was shown by Brandstädt et al. [9]. Bonsma [8] proved that this problem is NP-complete even for planar graphs with girth 5 and for planar graphs with maximum degree 4.

The Skew Partition problem is that of testing whether a graph $G = (V, E)$ has a disconnected cut U so that $V \setminus U$ induces a disconnected graph in the complement of G. De Figueiredo, Klein, Kohayakawa and Reed [24] proved that even the list version of this problem, where each vertex has been assigned a list of blocks in which it must be placed, is polynomial-time solvable. Afterwards, Kennedy and Reed [64] gave a faster polynomial-time algorithm for the non-list version.

Finally, for an integer $k \geq 1$, a cut U of a connected graph G is a k-cut of G if $G[U]$ contains exactly k components. For $k \geq 1$ and $\ell \geq 2$, a k-cut U is a (k, ℓ)-cut of a graph G if $G[V \setminus U]$ consists of exactly ℓ components. Ito et al. [55] proved that testing if a graph has a k-cut is solvable in polynomial time for $k = 1$ and NP-complete for every fixed $k \geq 2$. In addition they showed that testing if a graph has a (k, ℓ)-cut is polynomial-time solvable if $k = 1, \ell \geq 2$ and NP-complete otherwise [55]. The same authors showed, by using the approach for solving Disconnected Cut on planar graphs, that both problems are polynomial-time solvable on planar graphs.

3.2 Preliminaries

Let $G = (V, E)$ be a graph. We recall the following operations. The *contraction* of an edge uv removes u and v from G, and replaces them by a new vertex made adjacent to precisely those vertices that were adjacent to u or v in G. Unless we explicitly say otherwise we remove all self-loops and multiple edges so that the resulting graph stays simple. The *subdivision* of an edge uv replaces uv by a new vertex w with edges uw and vw. Let $u \in V$ be a vertex that has exactly two neighbours v, w, and moreover let v and w be non-adjacent. The *vertex dissolution* of u removes u and adds the edge vw.

A graph G contains a graph H as a *minor* if H can be obtained from G by a sequence of vertex deletions, edge deletions and edge contractions. If G does not contain H as a minor, G is *H-minor-free*. We say that G contains H as a *contraction*, denoted by $H \leq_c G$, if H can be obtained from G by a sequence of edge contractions. Finally, G contains H as a *subdivision* if H can be obtained from G by a sequence of vertex deletions, edge deletions and vertex dissolutions, or equivalently, if G contains a subgraph H' that is a *subdivision* of H, that is, H can be obtained from H' after applying zero or more vertex dissolutions. We say that a vertex in H' is a *subdivision vertex* if we need to dissolve it in order to obtain H; otherwise it is called a *branch vertex* (that is, it corresponds to a vertex of H). We note that some subdivisions may be reachable in different ways. For example P_3 is a subdivision of P_4 with the end vertices being branch vertices, however, either one of the inner vertices could be the subdivision vertex. When we refer to branch vertices and subdivision vertices, we do so with respect to a particular subgraph H' and a particular set of vertex dissolutions. In particular, in this thesis we are only interested in D_r as a subdivision for some r. As such, branch vertices have degree 4 while subdivision vertices have degree 2 so there is only one way to dissolve the vertices of a given H' to obtain D_r.

For some of our proofs the following global structure is useful. Let $G = (V_G, E_G)$ and $H = (V_H, E_H)$ be two graphs. An *H-witness structure* $\mathcal{W}$ is a vertex partition of a (not necessarily proper) subgraph of G into $|V_H|$ nonempty sets $\{W(x)\}_{x \in V_H}$ called *(H-witness) bags*, such that

(i) each $W(x)$ induces a connected subgraph of G,

(ii) for all $x, y \in V_H$ with $x \neq y$, bags $W(x)$ and $W(y)$ are adjacent in G if x and y are adjacent in H.

In addition, we may require the following additional conditions:

(iii) for all $x, y \in V_H$ with $x \neq y$, bags $W(x)$ and $W(y)$ are adjacent in G *only* if x and y are adjacent in H,

(iv) every vertex of G belongs to some bag.

By contracting all bags to singletons we observe that H is a minor or contraction of G if and only if G has an H-witness structure such that conditions (i)-(ii) or (i)-(iv) hold, respectively. We note that G may have more than one H-witness structure with respect to the same containment relation.

3.3 3-Connected Planar Graphs

We first present a necessary and sufficient condition for a 3-connected $K_{3,3}$-minor-free graph to have a minimal disconnected cut. We recall that we say $H \leq_c G$ when H is a contract of G.

Theorem 3.1. *A 3-connected $K_{3,3}$-minor-free graph G has a minimal disconnected cut if and only if $K_{2,r} \leq_c G$ for some $r \geq 2$.*

Proof. Let $G = (V, E)$ be a 3-connected graph that has no $K_{3,3}$ as a minor. First suppose that G has a minimal disconnected cut U. Let p and q be the number of components of $G[U]$ and $G[V \setminus U]$, respectively. Because U is a disconnected cut, $p \geq 2$ and $q \geq 2$. By definition, every vertex of every component of $G[U]$ is adjacent to all components in $G[V \setminus U]$. Hence, G contains $K_{p,q}$ as a contraction. Because G has no $K_{3,3}$ as a minor, G has no $K_{3,3}$ as a contraction. This means that $p \leq 2$ or $q \leq 2$. Because $p \geq 2$ and $q \geq 2$ holds as well, we find that $K_{2,r} \leq_c G$ for some $r \geq 2$.

Now suppose that $K_{2,r} \leq_c G$ for some $r \geq 2$. Throughout the remainder of the proof we denote the partition classes of $K_{k,\ell}$ by $X = \{x_1, \ldots, x_k\}$ and $Y = \{y_1, \ldots, y_\ell\}$. We refer to the bags in a $K_{k,\ell}$-witness structure of G corresponding to the vertices in X and Y as x-bags and y-bags, respectively. Because $K_{2,r} \leq_c G$, there exists a $K_{2,r}$-witness structure $\mathcal{W}$ of G that satisfies conditions (i)-(iv). Note that $W(x_1) \cup W(x_2)$ is a disconnected cut. However, it may not be minimal.

Suppose that $W(x_1)$ contains a vertex u that is adjacent to some but not all y-bags, i.e., the number of y-bags to which u is adjacent is h for some $1 \leq h < r$. Then we move u to a y-bag that contains one of its neighbours unless $W(x_1) \cup W(x_2)$ (without u) no longer induces a disconnected graph (which will be the case if u is the only vertex in $W(x_1)$). We observe that $G[W(x_1) \setminus \{u\}]$

may be disconnected, namely when u is a cut vertex in $G[W(x_1)]$. We also observe that u together with its adjacent y-bags induces a connected subgraph of G. Hence, the resulting witness structure $\mathcal{W}'$ (after moving vertices out of $W(x_1)$) is a $K_{q,r'}$-witness structure of G with $q \geq 2$ (as the resulting vertices in $W(x_1) \cup W(x_2)$ still induce a disconnected graph) and $r' = r - (h-1)$. Because $1 \leq h < r$, we find that $2 \leq r' \leq r$. We repeat this rule as long as possible. During this process, $W(x_2)$ does not change, and afterwards, we do the same for $W(x_2)$. Let $\mathcal{W}^*$ denote the resulting witness structure that is a K_{q^*,r^*}-witness structure satisfying conditions (i)-(iv) for some $q^* \geq 2$ and $2 \leq r^* \leq r$.

We will now prove the following claim.

Claim 3.1. *Every vertex of each x-bag of $\mathcal{W}^*$ is adjacent to all y-bags.*

We prove this claim as follows. First suppose that there exists an x-bag of $\mathcal{W}^*$, say $W^*(x_1)$, that contains a vertex u adjacent to some but not to all y-bags of $\mathcal{W}^*$, say u is not adjacent to $W^*(y_1)$. By our procedure we would have moved u to an adjacent y-bag unless that makes the disconnected cut connected. Hence we find that there are exactly two witness bags $W^*(x_1)$ and $W^*(x_2)$ and that $W^*(x_1) = \{u\}$. In our procedure we only moved vertices from x-bags to y-bags. This means that u belonged to an x-bag of the original witness structure $\mathcal{W}$. This x-bag was adjacent to all y-bags of $\mathcal{W}$ (as $\mathcal{W}$ was a $K_{2,r}$-witness structure). As we only moved vertices from x-bags to y-bags, this means that there must still exist a path from u to a vertex in $W^*(y_1)$ that does not use any vertex of $W^*(x_2)$; a contradiction. Hence every x-bag of $\mathcal{W}^*$ only contains vertices that are either adjacent to all y-bags or to none of them.

Now, in order to obtain a contradiction, suppose that an x-bag, say $W^*(x_1)$, contains a vertex u not adjacent to any y-bag. Because G is 3-connected, G contains three vertex-disjoints paths P_1, P_2, P_3 from u to a vertex in $W^*(y_1)$ (by Menger's Theorem). Each P_i contains a vertex v_i in $W^*(x_1)$ whose successor on P_i is outside $W^*(x_1)$ and thus in some y-bag. Hence, by our assumption, v_i

has a neighbour in every y-bag (including $W^*(y_1)$). Recall that the number of y-bags is $r^* \geq 2$. We consider the subgraph induced by the vertices from $W^*(y_1)$ and $W^*(y_2)$ together with the vertices on the three paths P_1, P_2, P_3. For $i = 1, \ldots, 3$ we contract all edges on the subpath of P_i from u to v_i to one edge, and we contract both $W^*(y_1)$ and $W^*(y_2)$ to single vertices. These edge contractions modify the subgraph into a graph isomorphic to $K_{3,3}$, which is not possible. Hence, every vertex of each x-bag of $\mathcal{W}^*$ is adjacent to all y-bags. This completes the proof of Claim 3.1.

As $q^* \geq 2$ and $r^* \geq 2$, there are at least two x-bags and at least two y-bags in $\mathcal{W}^*$. By combing this observation with Claim 3.1, we find that the x-bags of $\mathcal{W}^*$ form a minimal disconnected cut U of G. This completes the proof of Theorem 3.1. □

Recall that planar graphs are $K_{3,3}$-minor-free by Kuratowski's Theorem. Hence, by Theorem 3.1, in order to show that a 3-connected planar graph has a minimal disconnected cut, it suffices to find a $K_{2,r}$-contraction for some $r \geq 2$. Below we state some additional terminology.

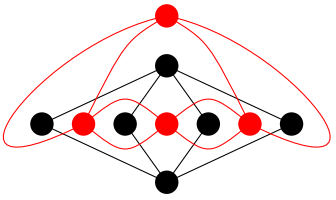


Figure 3.3: The graph $K_{2,4}$ in black and its dual in red.

Recall that D_n is the multigraph obtained from the cycle on $n \geq 3$ vertices by doubling its edges. We let D_2 be the multigraph that has two vertices with four edges between them. The *dual* graph G_d of a plane graph G has a vertex for each face of G, and there exist k edges between two vertices u and v in G_d if and only if the two corresponding faces share k edges in G. Note that the dual of a graph may be a multigraph. As 3-connected planar graphs have a unique embedding (see e.g. Lemma 2.5.1, p.39 of [81]) we can speak of the dual of a 3-connected planar graph.

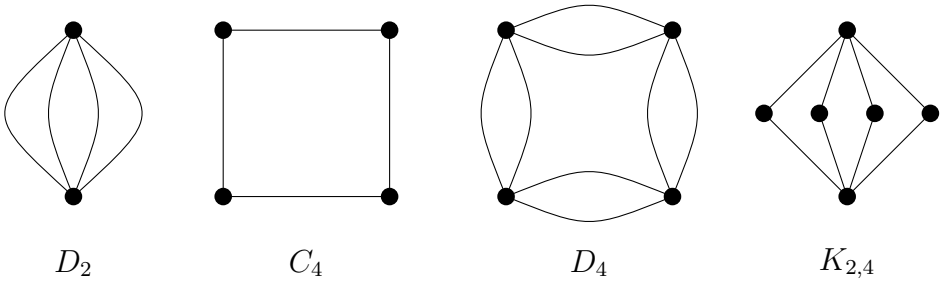


Figure 3.4: The graphs $D_2, C_4, D_4, K_{2,4}$. Note that the dual of $C_4 = K_{2,2}$ is D_2, that D_4 is obtained from C_4 by duplicating each edge and that D_4 is the dual of $K_{2,4}$.

Lemma 3.1. *Let G be a 3-connected planar graph. Then G contains $K_{2,r}$ as a contraction for some $r \geq 2$ if and only if the dual of G contains D_r as a subdivision.*

Proof. We first observe that for all $r \geq 2$, every $K_{2,r}$ has a unique plane embedding, the dual of which is D_r. Then the results follows from a result from [62] that for 3-connected planar graphs comes down to the following statement: a 3-connected planar graph G contains a graph H as a contraction if and only if the dual of G contains the dual of H as a subdivision. □

By Lemma 3.1 it suffices to check if the dual of the 3-connected planar input graph contains D_r as a subdivision for some $r \geq 2$. We show how to solve this problem in polynomial time for general graphs. In order to do so we need the next lemma which gives a sufficient condition for a graph G to be a yes-instance of this problem. In its proof we use the following notation. For a path $P = v_1 v_2 \dots v_p$, we write $v_i P v_j$ to denote the subpath $v_i v_{i+1} \dots v_j$ or $v_j P v_i$ if we want to emphasise that the subpath is to be traversed from v_j to v_i.

Lemma 3.2. *Let v, w be two distinct vertices of a multigraph G such that there exist four edge-disjoint v-w-paths in G. Then G contains a subdivision of D_r for some $r \geq 2$.*

Proof. We prove the lemma by induction on $|V_G| + |E_G|$. Then we can assume that G is the union of the four edge-disjoint v-w-paths. Let us call these paths P_1, P_2, P_3, and P_4. If these four paths are vertex-disjoint (apart from v and w) then they form a subdivision of D_2. Hence, we may assume that there exists at least one vertex of G not equal to v or w that belongs to more than one of the four paths.

First suppose that there exists a vertex u that belongs to all four paths P_1, P_2, P_3 and P_4. Let $G' = (V_{G'}, E_{G'})$ be the graph consisting of the vertices and edges of the four subpaths vP_1u, vP_2u, vP_3u and vP_4u. As G' does not contain w, it holds that $V_{G'} + E_{G'} < |V_G| + |E_G|$. By the induction hypothesis, G', and thus G, contains a subdivision of D_r for some $r \geq 2$.

Now suppose that there exists a vertex u that belong to only three of the four paths, say to P_1, P_2, and P_3. Let G' be the graph that consists of the vertices and edges of the four paths uP_1w, uP_2w, uP_3w and uP_1vP_4w. As G' does not contain an edge of vP_2u we find that $V_{G'} + E_{G'} < |V_G| + |E_G|$. By the induction hypothesis, G', and thus G, contains a subdivision of D_r for some $r \geq 2$.

From now on assume that every inner vertex of every path P_i $(i = 1, \ldots, 4)$ belongs to at most one other path P_j $(j \neq i)$. We say that two different paths P_i and P_j *cross* in a vertex u if u is an inner vertex of both P_i and P_j. Suppose P_i and P_j cross in some other vertex u' as well. Then we say that u is *crossed before* u' by P_i and P_j if u is an inner vertex of both vP_iu' and vP_ju'.

We now prove the following claim.

Claim 3.2. *If P_i and P_j $(i \neq j)$ cross in both u and u' then we may assume without loss of generality that either u is crossed before u' or u' is crossed before u.*

We prove Claim 3.2 as follows. Suppose that u is not crossed before u' by P_i and P_j and similarly that u' is not crossed before u by P_i and P_j. Then we may assume without loss of generality that u is an inner vertex of vP_iu' and that u' is

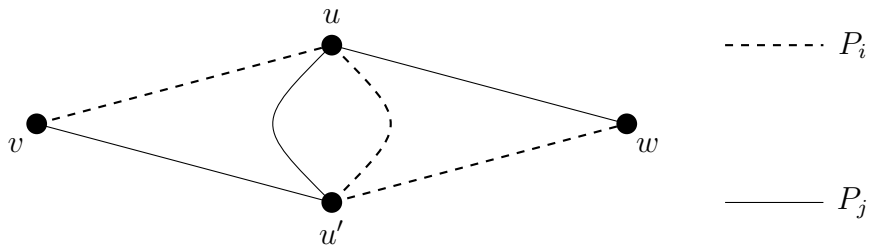


Figure 3.5: The paths P_i and P_j where u is not crossed before u' by P_i and P_j and similarly u' is not crossed before u by P_j and P_i. Note that the paths P_i and P_j may have more common vertices, but for clarify this is not been shown.

an inner vertex of vP_ju. See Figure 3.5 for an example of this situation. However, in that case we can replace P_i and P_j by the paths vP_iuP_jw and $vP_ju'P_iw$. These two paths together with the two unused original paths form a subgraph G' of G with fewer edges than G (as for instance no edge on uP_iu' belongs to G'). We apply the induction hypothesis on G'. This completes the proof of Claim 3.2.

We need Claim 3.2 to prove the following claim, which is crucial for our proof.

Claim 3.3. *We may assume without loss of generality that there exists a vertex* $u \notin \{v, w\}$ *that is on two paths* P_i *and* P_j $(i \neq j)$ *so that every inner vertex of* vP_iu *and* vP_ju *has degree* 2 *in* G.

We prove Claim 3.3 as follows. By our assumption there exists at least one vertex in G that is on two paths. Let $s \notin \{v, w\}$ be such a vertex, without loss of generality let s belong to P_1. Suppose that there is an inner vertex of vP_1s with degree larger than 2, we may simply choose this vertex to be s instead and as such we may assume that every inner vertex of vP_1s has degree 2. Since s has degree larger than 2, it belongs to another path, without loss of generality say it belongs to P_2. Then, by Claim 3.2, we find that P_1 and P_2 do not cross in an inner vertex of vP_2s.

If every inner vertex of vP_1s and vP_2s has degree 2 in G then the claim has been proven. Suppose otherwise, namely that there exists an inner vertex s' of vP_1s or vP_2s whose degree in G is larger than 2, say s' belongs to vP_2s. As P_1 does

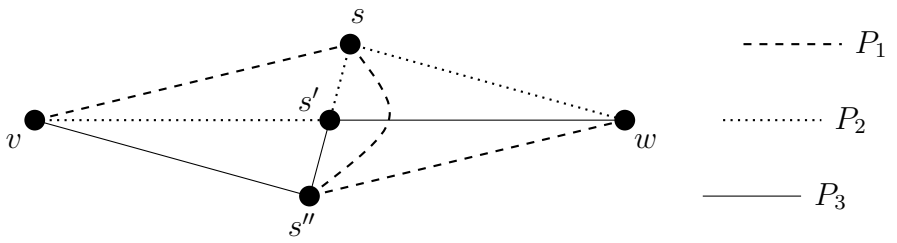


Figure 3.6: The paths P_1, P_2 and P_3 where s belongs to P_1 and P_2, s' belongs to vP_2s and P_3 and s'' belongs to vP_3s' and P_1.

not cross vP_2s, we find that s' must belong to P_3 or to P_4. Choose s' in such a way that every inner vertex of vP_2s' has degree 2 in G. Assume without loss of generality that s' belongs to P_3.

If every inner vertex of vP_3s' has degree 2 then the claim has been proven (as every inner vertex of vP_2s' has degree 2 as well). Suppose otherwise, namely that there exists an inner vertex s'' of vP_3s' whose degree in G is larger than 2. Choose s'' in such a way that every inner vertex of vP_3s'' has degree 2 in G. By Claim 3.2, no inner vertex of vP_3s' belongs to P_2, so s'' does not lie on P_2. This means that s'' belongs either to P_1 or to P_4.

Suppose s'' belongs to P_1. See Figure 3.6 for an example of this situation. As every inner vertex of vP_1s has degree 2, we find that s is an inner vertex of vP_1s''. However, we can now replace P_1, P_2 and P_3 by the three paths $vP_1sP_2w, vP_2s'P_3w$ and $vP_3s''P_1w$. These three paths form, together with P_4, a subgraph of G with fewer edges than G (for instance, no edge of sP_1s'' belongs to G'). We can apply the induction hypothesis on this subgraph. Hence we may assume that s'' does not belong to P_1.

From the above we conclude that s'' belongs to P_4. See Figure 3.7 for an example of this situation. We consider the paths vP_3s'' and vP_4s''. If every inner vertex of vP_4s'' has degree 2 in G then we have proven Claim 3.3 (recall that every inner vertex of vP_3s'' has degree 2 in G as well). Suppose otherwise, namely that there exists an inner vertex t of vP_4s'' whose degree in G is larger than 2. Choose t in such a way that every inner vertex of vP_4t has degree 2 in G. By Claim 3.2

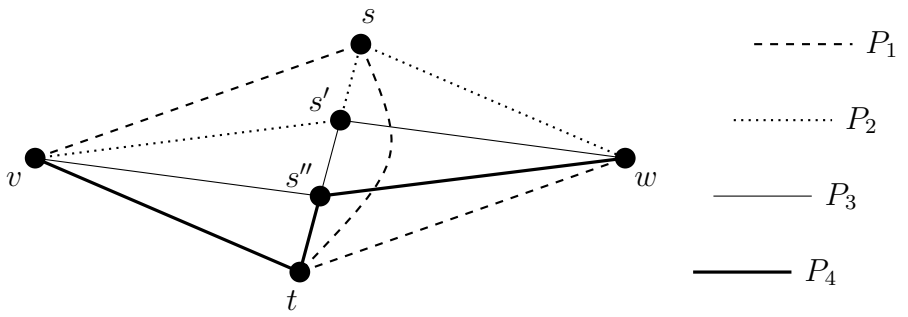


Figure 3.7: The paths P_1, P_2, P_3 and P_4 where s belongs to P_1 and P_2, s' belongs to vP_2s and P_3, s'' belongs to vP_3s' and P_4 and t belongs to vP_4s'' and P_1.

we find that t is not on P_3. If t is on P_2 we can use a similar replacement of three paths by three new paths as before that enables us to apply the induction hypothesis. Hence, we find that t belongs to P_1.

As every inner vertex of vP_1s has degree 2 in G we find that s is an inner vertex of vP_1t. Then we take the four paths $vP_1sP_2w, vP_2s'P_3w, vP_3s''P_4w$ and vP_4tP_1w. These four paths form a subgraph G' of G with fewer edges than G (as for instance G' contains no edge from sP_1t). We can apply the induction hypothesis on G'. Hence we may assume that such a vertex t cannot exist. Thus we have found the desired vertex and subpaths, namely s'' with subpaths vP_3s'' and vP_4s''. This completes the proof of Claim 3.3.

By Claim 3.3 we may assume without loss of generality that there exists a vertex u that belongs to P_1 and P_2 such that every inner vertex of vP_1u and vP_2u has degree 2. Let G^* be the graph obtained from G by contracting all edges of vP_1u and vP_2u (recall that we remove loops and multiple edges). Let u^* be the new vertex to which all the edges were contracted. Notice that there are four edge-disjoint u^*-w-paths in G^*. Then, by the induction hypothesis, G^* contains a subdivision H of D_r for some $r \geq 2$. If u^* does not belong to H, then G contains H as well and we would have proven the lemma. Assume that u^* belongs to H.

First suppose that u^* is a subdivision vertex of H in G^*. Let u^* have neighbours s_1 and s_2 in H. Take a shortest path Q from s_1 to s_2 in the subgraph of G induced

by s_1, s_2 and the vertices of vP_1u and vP_2u. This results in a graph H', which is a subgraph of G and which is a subdivision of D_r as well.

Now suppose that u^* is a branch vertex of H in G^*, say u^* corresponds to $z \in V_{D_r}$. Note that any vertex in D_r has one neighbour if $r = 2$ and two neighbours if $r \geq 3$. We let s and t be the two branch vertices of H that correspond to the neighbours of z in D_r (note that $s = t$ if $r = 2$). Let s_1 and s_2 be the neighbours of u^* on the two paths from u^* to s, respectively, in H. Similarly, let t_1 and t_2 be the neighbours of u^* on the two paths from u^* to t, respectively, in H. Note that, as G is a multigraph, it is possible that $s_1 = s_2 = s$ and $t_1 = t_2 = t$.

Recall that every internal vertex on vP_1u and on vP_2u has degree 2 in G. As u is an inner vertex of P_1 and P_2 but not of P_3 and P_4, it has degree 4 in G. As G is the union of P_1, P_2, P_3 and P_4, we find that v has degree 4 as well. Then, after uncontracting u^*, we have without loss of generality one of the following two situations in G. First, u is adjacent to s_1 and s_2 and v is adjacent to t_1 and t_2. In that case u and v become branch vertices of a subdivision of D_{r+1} in G (to which the internal vertices on the paths uP_1v and uP_2v belong as well, namely as subdivision vertices). Second, u is adjacent to s_1 and t_1, whereas v is adjacent to s_2 and t_2. Then u and v become subdivision vertices of a subdivision of D_r in G (and we do not use the internal vertices on the paths uP_1v and uP_2v). This completes the proof of the lemma. □

Lemma 3.2 gives us the following result.

Theorem 3.2. *It is possible to find in $\mathcal{O}(mn^2)$ time whether a graph G with n vertices and m edges contains D_r as a subdivision for some $r \geq 2$.*

Proof. Let G be a graph with m edges. We check for every pair of vertices s and t whether G contains four edge-disjoint paths between them. We can do this via a standard reduction to the maximum flow problem. Replace each edge uv by the arcs (u, v) and (v, u). Give each arc capacity 1. Introduce a new vertex

s' and an arc (s', s) with capacity 4. Also introduce a new vertex t' and an arc (t, t') with capacity 4. Check if there exists an (s', t')-flow of value 4 by using the Ford-Fulkerson algorithm. As the maximum value of an (s', t')-flow is at most 4, this costs $\mathcal{O}(m)$ time per pair, so $\mathcal{O}(mn^2)$ time in total.

If there exists a pair s, t in G with four edge-disjoint paths between them then G has a subdivision of D_r, for some $r \geq 2$, by Lemma 3.2. If not then we find that G has no subdivision of any D_r ($r \geq 2$) as any subdivision of D_r immediately yields four edge-disjoint paths between two vertices and our algorithm would have detected this. □

We are now ready to state our main result.

Theorem 3.3. Minimal Disconnected Cut *can be solved in* $\mathcal{O}(n^3)$ *time on* 3*-connected planar graphs with* n *vertices.*

Proof. Let G be a 3-connected planar graph with n vertices. By Theorem 3.1 it suffices to check whether $K_{2,r} \leq_c G$ for some $r \geq 2$. By Lemma 3.1, the latter is equivalent to checking whether the dual of G, which we denote by G^*, contains D_r as a subdivision for some $r \geq 2$. To find G^* we first embed G in the plane using the linear-time algorithm from Mohar [80]. As the number of edges in a planar graph is linear in the number of vertices, G^* has $\mathcal{O}(n)$ vertices and $\mathcal{O}(n)$ edges and can be constructed in $\mathcal{O}(n)$ time. We are left to apply Theorem 3.2. □

3.4 2-Connected Planar Graphs

We prove the following result, which shows that Theorem 3.3 cannot be generalised to any further class of k-connected planar graphs.

Theorem 3.4. Minimal Disconnected Cut *is* NP*-complete for the class of* 2*-connected planar graphs.*

Proof. As we can check in polynomial time whether a given subset of vertices in a graph is a minimal disconnected cut, the problem belongs to NP. To see this we observe that we can determine whether a graph G is connected or not in $\mathcal{O}(n+m)$ time using either a breath-first search of depth-first search. Given a set S, we can easily check that both $G[S]$ and $G[V_G \setminus S]$ are disconnected. It then remains to show that S is minimal, we can do this be checking that $G[(V_G \setminus S) + v]$ is connected for every $v \in S$. To show NP-hardness we reduce from Stable Cut. Recall that this problem is to test whether a graph has a cut that is an independent set and that it is an NP-complete problem for planar graphs [68] even if they are 2-connected (as the answer is trivially yes if the input graph contains a cut vertex[1]).

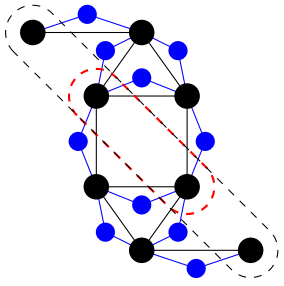


Figure 3.8: The graph G' with the vertices and edges of G shown in black. A stable cut and a minimal stable cut are shown with black and red dashed lines respectively.

Let $G = (V_G, E_G)$ be a 2-connected planar graph with n vertices and m edges. We construct in polynomial time a graph G' by adding for each edge $e = uv$ in G

[1]We recall from Section 3.1 that Stable Cut is NP-complete even for 3-connected planar graphs, but we do not need 3-connectivity in our proof.

a new (blue) vertex x_e that we make adjacent (only) to u and v. Note that G' is a planar graph with $m + n$ vertices and $3m$ edges. Moreover, G' is 2-connected. Hence, it suffices to prove that G has a stable cut if and only if G' has a minimal disconnected cut. We see an example of this in Figure 3.8.

First suppose that G has a stable cut S. As long as S contains a vertex u so that the subgraph of G induced by $(V_G \setminus S) \cup \{u\}$ is disconnected we move u from S to $V_G \setminus S$. Because G is 2-connected, the resulting set $S^* \subseteq S$ is a stable cut of size at least 2. By our procedure, S^* is a minimal disconnected cut of G as well. Because S^* is an independent set, at least one vertex of every pair of adjacent vertices u, v in G does not belong to S^*, say u does not belong to S. Let F be the component of $G[V \setminus S^*]$ that contains u. Then either v belongs to F as well or v belongs to S^*. In both cases we place x_{uv} in F (so we neither create any new components in $G^* - S$ nor do we reduce the number of components). After doing this for each pair of adjacent vertices in G we find that S^* is also a minimal disconnected cut of G^*.

Now suppose that G' has a minimal disconnected cut S'. Consider an edge uv of G' that belongs to G as well. We see that the vertex x_{uv} (a blue vertex in Figure 3.8) can not belong to a *minimal* cut since its only neighbours are adjacent and hence belong to at most one component of $G' - S'$. If x_{uv} were in S' we would see that $S' - x_{uv}$ is also a cut set and as such S' would not be minimal. Hence, x_{uv} cannot belong to a *minimal* cut since it's only neighbours are adjacent and hence can belong to at most one component. Moreover, at most one of u and v can belong to S' as otherwise, due to their adjacency, they would belong to the same component of $G'[S']$, meaning that any other component of $G'[S']$ is not adjacent to the 1-vertex component of $G' - S'$ that contains x_{uv}. Hence, S' is a stable cut of G' that only contains vertices of G. Because at least one vertex of any pair of adjacent vertices u, v belongs to the same component of $G' - S'$ that contains the vertex x_{uv}, we find that $G - S'$ has just as many (and thus at

least two) components as $G' - S'$. We conclude that S' is a stable cut of G as well. □

3.4.1 Minimal Connected Cut on 2-Connected Apex Graphs

We recall the a graph is an *apex graph* if it can be made planar by the deletion of a single vertex. As such we see that the class of apex graphs includes the class of planar graphs.

A cut S in a graph G is a *minimal connected* cut if $G[S]$ is connected and for all $u \in S$ we have that $G[(V \setminus S) \cup \{u\}]$ is connected. We call the problem of testing whether a graph has a minimal connected cut of size at least k the Minimal Connected Cut(k) problem. By modifying the proof of Theorem 3.4 we obtain the following result.

Theorem 3.5. Minimal Connected Cut(3) *is* NP*-complete even for the class of* 2*-connected apex graphs.*

Proof. We can check in polynomial time (see Theorem 3.4) whether a given subset of vertices in a graph is a minimal connected cut. Hence the problem belongs to NP. As mentioned, we are following the line of the proof of Theorem 3.4, so we reduce from the Stable Cut problem restricted to 2-connected planar graphs.

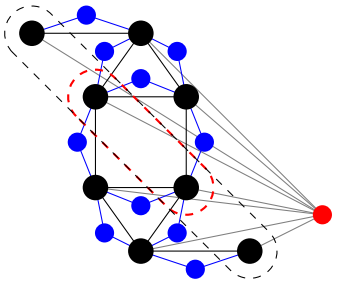


Figure 3.9: The graph G'' with the vertices and edges of G shown in black and with y in red. A stable cut and a minimal stable cut of G are shown with black and red dashed lines respectively.

Let G be a 2-connected planar graph with n vertices and m edges. We construct in polynomial time a graph G'' by adding for each edge $e = uv$ in G a new (blue)

vertex x_e that we make adjacent (only) to u and v. We say that these newly added vertices are of *x-type*. Afterwards we add a new vertex y that we make adjacent to all vertices of G (so not to the x-type vertices). Note that G'' is an apex graph with $m+n+1$ vertices and $3m+n$ edges. We see an example of this in Figure 3.9 Moreover, G'' is 2-connected. We claim that G has a stable cut if and only if G'' has a minimal connected cut of size at least 3.

First suppose that G has a stable cut S. Following the same arguments as in the proof of Theorem 3.4 we find that S contains a subset S^* that is a minimal disconnected cut of $G'' - \{y\}$. Adding y to S^* yields a minimal connected cut of G''. Because G is 2-connected, $S^* \cup \{y\}$ has size at least 3.

Now suppose that G'' has a minimal connected cut S'' of size at least 3. Consider an edge uv of G'' that belongs to G as well. We see that the vertex x_{uv} (a blue vertex in Figure 3.9) can not belong to a *minimal* cut since its only neighbours are adjacent and hence belong to at most one component of $G' - S'$. If x_{uv} were in S'' we would see that $S'' - x_{uv}$ is also a cut set and as such S'' would not be minimal. Moreover, at most one of u and v can belong to S''. This can be seen as follows. For contradiction, assume that u and v both belong to S''. Because S'' has size at least 3, we find that S'' contains some vertex $w \notin \{u, v\}$. This is not possible, as w is not adjacent to the 1-vertex component of $G'' - S''$ that contains x_{uv}.

Let $T = S'' \setminus \{y\}$ if $y \in S''$ and let $T = S''$ otherwise. Because at most one of every pair of adjacent vertices in G and no x-type vertices belong to S'', we find that T is a stable cut of $G'' - \{y\}$ that only contains vertices of G. Because at least one vertex of any pair of adjacent vertices u, v belongs to the same component of $G'' - S''$ that contains the vertex x_{uv}, we find that $G - T$ has just as many (and thus at least two) components as $G'' - S''$. Hence $G - T$ has at least two components. We conclude that T is a stable cut of G. □

Note that we cannot use the reduction in the proof of Theorem 3.5 to get NP-hardness for Minimal Connected Cut(1), the reason being that every edge $uv \in G$ is a minimal connected cut of size 2 in G'' as when removed it disconnects x_{uv} from the remainder of the graph.

3.5 A Generalisation

Let $\mathcal{H}$ be a graph class. Recall that a given connected graph has a minimal $\mathcal{H}$-cut if it has a (minimal) cut that induces a graph in $\mathcal{H}$ and that the corresponding decision problems are called $\mathcal{H}$-Cut and Minimal $\mathcal{H}$-Cut.

From the proof of Theorem 3.4 we find that the Minimal Stable Cut problem, that is, the problem of determining whether a graph has a minimal stable cut is NP-complete even for 2-connected planar graphs. The argument in this proof to move any cut vertex not adjacent to all components outside the stable cut until a minimal stable cut is obtained can be generalised to $\mathcal{H}$-cuts if an extra condition is added.

Observation 3.1. *Let $\mathcal{H}$ be a graph class of graphs closed under vertex deletion. Then a connected graph has a minimal $\mathcal{H}$-cut if and only if it has a $\mathcal{H}$-cut.*

Due to Observation 3.1, the problems $\mathcal{H}$-Cut and Minimal $\mathcal{H}$-Cut are polynomially equivalent if $\mathcal{H}$ is closed under vertex deletion. Recall that if we let $\mathcal{H}$ be the class of disconnected graphs, we obtain the (Minimal) Disconnected Cut problem. However, we cannot combine Observation 3.1 with results for the Disconnected Cut problem to obtain corresponding results for the Minimal Disconnected Cut problem, because the class of disconnected graphs is not closed under vertex deletion. This is also clear from the fact that Disconnected Cut is polynomial-time solvable for planar graphs [55], whereas we showed in Section 3.4 that Minimal Disconnected Cut is NP-complete even for 2-connected planar graphs.

Also if for instance $\mathcal{H}$ consists of all linear forests on at least two components (disjoint unions of two or more paths) we cannot use Observation 3.1, but in that case we can determine the complexity of Minimal $\mathcal{H}$-Cut by first giving the following description of minimal disconnected cuts in planar graphs (the second

statement is a structural observation which is not needed for the proof of this result).

Theorem 3.6. *Let G be a $K_{3,3}$-minor-free graph. Let U be any minimal cut of G. Then every component of $G[U]$ is a path or a cycle, or in case G is planar and U is disconnected, every component of $G[U]$ is a path. Moreover, G has a minimal disconnected cut of size 2 or for every minimal disconnected cut U of G it holds that $G[V \setminus U]$ has exactly two components.*

Proof. Let V_1 and V_2 be the vertex sets of any two components of $G[V \setminus U]$. Suppose that U contains a vertex s of degree 3 in $G[U]$. Then s has neighbours t_1, t_2, t_3 in U. As every vertex of U is adjacent to both V_1 and V_2, the vertices s, t_1, t_2, t_3 form, together with V_1 and V_2, a $K_{3,3}$-minor of G, a contradiction. Hence, every component of $G[U]$ has maximum degree at most 2, so is either a path or a cycle. Suppose that G is planar and that U is disconnected. For contradiction, assume that $G[U]$ contains a component with vertex set U_1 that is a cycle. As every vertex of U is adjacent to both V_1 and V_2 we find that U_1 and a vertex of another component of $G[U]$ form, together with V_1 and V_2, a K_5-minor of G, which is not possible as G is planar. To see this contract V_1 and V_2 to single vertices and U_1 to a cycle of length 3.

Now suppose that G has at least one minimal disconnected cut but not one of size 2. Let U be a minimal disconnected cut of G. Then $G[V \setminus U]$ must have exactly two components; otherwise three vertices from U and three components of $G[V \setminus U]$ form a $K_{3,3}$-minor of G, as every vertex of U is adjacent to every component of $G[V \setminus U]$ by definition. □

Let $\mathcal{P}$ consist of all disjoint unions of two or more paths. Theorems 3.3, 3.4 and 3.6 have the following consequence.

Corollary 3.1. Minimal $\mathcal{P}$-Cut *is polynomial-time solvable for k-connected planar graphs if $k \geq 3$ and* NP*-complete if $k \leq 2$.*

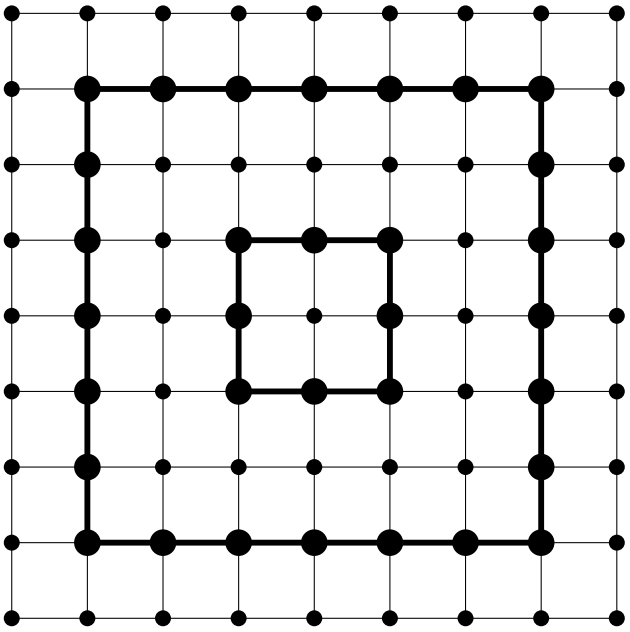


Figure 3.10: The grid M_9, where the two thick cycles correspond to cycles C_b and C_d in the proof of Theorem 3.7.

3.6 Semi-Minimality

The $m \times m$ *grid* M_m has all pairs (i, j) for $i, j = 0, 1, \ldots, m-1$ as the vertex set, and two vertices (i, j) and (i', j') are joined by an edge if and only if $|i-i'|+|j-j'| = 1$. See Figure 3.10 for an example. We need the following result due to Robertson, Seymour and Thomas.

Lemma 3.3 ([88])**.** *For every integer m, every planar graph of treewidth at least $6m - 4$ contains M_m as a minor.*

We also use the well-known result of Courcelle [20] that states that on any class of graphs of bounded treewidth, every problem definable in monadic second-order logic can be solved in time linear in the number of vertices of the graph (we refer to [20] for more details on monadic second-order logic).

Lemma 3.4. *The* Semi-Minimal Disconnected Cut *problem can be defined in monadic second order logic.*

Proof. We can express the property that $G = (V, E)$ has a semi-minimal disconnected cut in monadic second order logic as follows. We first note that a graph G has a disconnected cut if and only if V can be partitioned into four sets U_1, U_2, V_1, V_2 such that the following three conditions hold:

1. every vertex of V belongs to exactly one set of $\{U_1, U_2, V_1, V_2\}$;
2. sets U_1, U_2, V_1, V_2 are all nonempty;

3a. sets U_1 and U_2 are nonadjacent;

3b. sets V_1 and V_2 are nonadjacent.

It is readily seen that these conditions can be expressed in monadic second order logic:

- $\phi_1 = \forall u((\ \ U_1(u) \wedge \neg U_2(u) \wedge \neg V_1(u) \wedge \neg V_2(u))$
  $\vee(\neg U_1(u) \wedge \ \ U_2(u) \wedge \neg V_1(u) \wedge \neg V_2(u))$
  $\vee(\neg U_1(u) \wedge \neg U_2(u) \wedge \ \ V_1(u) \wedge \neg V_2(u))$
  $\vee(\neg U_1(u) \wedge \neg U_2(u) \wedge \neg V_1(u) \wedge \ \ V_2(u)))$;
- $\phi_2 = \exists u U_1(u) \wedge \exists u U_2(u) \wedge \exists u V_1(u) \wedge \exists u V_2(u)$;
- $\phi_{3a} = \forall u \forall v((U_1(u) \wedge U_2(v)) \rightarrow \neg E(u, v))$;
- $\phi_{3b} = \forall u \forall v((V_1(u) \wedge V_2(v)) \rightarrow \neg E(u, v))$.

We are left to express the semi-minimality in monadic second order logic. This condition is equivalent to demanding that for all $u \in U_1 \cup U_2$ there exists a set $Z_u \subseteq V$ (so Z_u may be different for different vertices u of $U_1 \cup U_2$) such that the following three conditions hold:

4a. $Z_u \cap (V_1 \cup V_2)$ contains a neighbour s of u;

4b. $(V_1 \cup V_2) \setminus Z_u$ contains a neighbour t of u;

4c. there is no edge between any vertex of $Z_u \cap (V_1 \cup V_2)$ and any vertex of $(V_1 \cup V_2) \setminus Z_u$.

Also these conditions can be easily formulated in monadic second order logic:

- $\phi_{4a} = \exists s(E(s, u) \wedge Z_u(s) \wedge (V_1(s) \vee V_2(s))$;
- $\phi_{4b} = \exists t(E(t, u) \wedge (V_1(t) \vee V_2(t)) \wedge \neg Z_u(t))$;

- $\phi_{4c} = \forall s \forall t((Z_u(s) \wedge (V_1(s) \vee V_2(s)) \wedge (V_1(t) \vee V_2(t)) \wedge \neg Z_u(t)) \rightarrow \neg E(s,t))$.

Then G has a semi-minimal disconnected cut if and only if the following monadic second order logic sentence is true:

$$\exists U_1 \exists U_2 \exists V_1 \exists V_2 (\phi_1 \wedge \phi_2 \wedge \phi_{3a} \wedge \phi_{3b} \wedge \forall u((U_1(u) \vee U_2(u)) \rightarrow \exists Z_u(\phi_{4a} \wedge \phi_{4b} \wedge \phi_{4c}))).$$

This completes the proof of Lemma 3.4. □

We also need the following lemma.

Lemma 3.5. *Let U be a disconnected cut of a connected graph $G = (V, E)$. If every component of $G[U]$ is adjacent to at least two components of $G[V \setminus U]$, then G has a semi-minimal disconnected cut $U' \subseteq U$.*

Proof. Let $D_1, \ldots, D_p$ be the components of $G[U]$. For each D_i we do as follows. As long as there exists a vertex $u \in D_i$ that is adjacent to at most one component of $G[V \setminus U]$ we move u from D_i to $G[V \setminus U]$. Because D_i is adjacent to at least two components of $G[V \setminus U]$, this process stops before D_i becomes empty. Afterwards, D_i only contains vertices adjacent to none or at least two components of $G[V \setminus U]$. Note that D_i contains at least one vertex adjacent to at least two components of $G[V \setminus U]$. We move all vertices not adjacent to any components of $G[V \setminus U]$ from D_i to $G[V \setminus U]$. Afterwards, D_i is still nonempty. Hence, after doing this for each D_i, we have obtained a set $U' \subseteq U$ that is a semi-minimal disconnected cut of G. □

We are now ready to prove the main result of this section.

Theorem 3.7. *The* Semi-Minimal Disconnected Cut *problem can be solved in linear time for planar graphs.*

Proof. Let G be a planar graph. We use Bodlaender's algorithm [7] to test in linear time whether the treewidth of G is at most $6 \cdot 9 - 5$. If so, then by Lemma 3.4

and the aforementioned theorem of Courcelle [20], we can test in linear time whether G has a semi-minimal disconnected cut. If not, then G contains M_9 as a minor by Lemma 3.3. Let $\mathcal{W}$ be a corresponding witness structure. We notice that the vertices of M_9 can be partitioned into 5 nested cycles C_a, C_b, C_c, C_d and C_e of length 1 (with slight abuse of terminology), 8, 16, 24 and 32, respectively; see also Figure 3.10. We let U be the union of vertices in the sets $W(x)$ for all $x \in V_{C_b} \cup V_{C_d}$. We now claim that U is a disconnected cut of G since G is planar and hence K_5-minor-free. First we show that $G[V_G \setminus U]$ is disconnected. Contract every vertex in $W(C_c)$ and $W(C_e)$ to two single vertices v_c and v_e respectively, and contract $W(C_d)$ to a cycle of length 3. We see that if $G[V_G \setminus U]$ were connected, there would be an edge between v_c and v_e, these 5 vertices would form K_5 as a minor. We may apply the same logic to show that $G[U]$ is disconnected by considering $W(C_b)$, $W(C_d)$ and $W(C_c)$. Moreover, as $V_{C_b} \cup V_{C_d}$ satisfies the condition of Lemma 3.5, U satisfies this condition as well. Hence, we can apply this lemma to obtain a semi-minimal disconnected cut $U' \subseteq U$ of G. □

3.7 Conclusions

Our main results are that Minimal Disconnected Cut is NP-complete for 2-connected planar graphs and polynomial-time solve for planar graphs that are 3-connected. Our proof technique for the latter result was based on translating the problem to a dual problem, namely the existence of a subdivision of D_r for *some* r, for which we obtained a polynomial-time algorithm even for general graphs. One can also solve the problem of determining whether a graph contains D_r as a subdivision for some *fixed* integer r by using the algorithm of Grohe, Kawarabayashi, Marx, and Wollan [48] which tests in cubic time, for any fixed graph H, whether a graph contains H as a subdivision. However, when r is part of the input we observe the following.

Theorem 3.8. *The problem of deciding whether a graph contains the graph D_r as a subdivision is* NP*-complete if r is part of the input.*

Proof. We reduce from the problem Hamiltonian Cycle, which is well known to be NP-complete [63]. Let G be a graph with n vertices and m vertices. We replace each edge $e = uv$ in G by two paths us_ev and ut_ev where s_e and t_e are two new vertices (so we add $2m$ new vertices in total). The resulting graph contains D_n as a subdivision if and only if G has a Hamiltonian cycle. □

We finish this chapter with some open problems. Recall that our construction in Theorem 3.5 does not work for proving NP-hardness of Minimal Connected Cut(1), which is the problem of deciding whether a graph has a minimal connected cut. The computational complexity of this problem is not known. In fact we do not know this even for (3-connected) planar graphs, and we pose these questions as open problems. Note that these problems fall under a more general study into minimal $\mathcal{H}$-cuts that we introduced in Section 3.5.

Square Root on Graphs with Low Clique Number

The *square* H^2 of a graph H has vertex set V_H and edge set $\{uv \mid \text{dist}_H(u,v) \leq 2\}$ as seen in Figure 4.1 below.

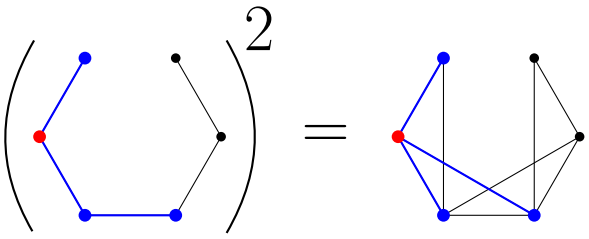


Figure 4.1: The graph P_6 and its square.

A graph H is a *square root* of a graph G if $H^2 = G$. There exist graphs with no square root, graphs with a unique square root and graphs with multiple square roots.

The Square Root problem is that of deciding whether a given graph admits a square root. By showing boundedness of treewidth we prove that Square Root is polynomial-time solvable on some classes of graphs with small clique number, in particular K_4-free graphs, 3-degenerate graphs and (K_r, P_t)-free graphs.

This result was initially presented at the 14th Cologne Twente Workshop (CTW 2016), Gargnano, Italy [44]. We have also presented related work at the 15th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2016), Reykjavik, Iceland [43] and at the 27th International Workshop on Combinatorial Algorithms (IWOCA 2016), Helsinki, Finland [42]. A journal paper containing the results presented at these conferences has been accepted by Discrete Applied Mathematics and we have submitted a further paper to Theory of Computing Systems.

4.1 Introduction

Squares and square roots are well-known concepts in graph theory that have been studied first from a structural perspective [83, 91] and later also from an algorithmic perspective (as we will discuss). The *square* $G = H^2$ of a graph $H = (V_H, E_H)$ is the graph with vertex set $V_G = V_H$, such that any two distinct vertices $u, v \in V_H$ are adjacent in G if and only if u and v are of distance at most 2 in H.

In this paper we study the reverse concept: a graph H is a *square root* of a graph G if $G = H^2$. There exist graphs with no square root (such as graphs with a cut-vertex), graphs with a unique square root (such as squares of cycles of length at least 7) as well as graphs with more than one square root (such as complete graphs).

In 1967 Mukhopadhyay [83] characterised the class of connected graphs with a square root. However, in 1994, Motwani and Sudan [82] showed that the decision problem Square Root, which asks whether a given graph admits a square root, is NP-complete.

Square Root	
Instance:	a graph G.
Question:	does there exist a graph H with $H^2 = G$?

Afterwards, Square Root was shown to be polynomial-time solvable for various graph classes, such as planar graphs [73], or more generally, any non-trivial minor-closed graph class [84]; line graphs [78]; trivially perfect graphs [79]; threshold graphs [79]; graphs of maximum degree 6 [16] and graphs of maximum average degree smaller than $\frac{46}{11}$ [43]. It was also shown that Square Root is NP-complete for chordal graphs [67]. We refer to [16, 17, 43] for a number of parameterised complexity results on Square Root.

For several well-known graph classes the complexity of Square Root is still unknown. For example, Milanic and Schaudt [79] posed the complexity of Square Root restricted to split graphs and cographs as open problems. In Table 4.1 we survey the known results.

graph class $\mathcal{G}$	complexity
planar graphs [73]	linear
non-trivial and minor-closed [84]	linear
K_4-free graphs*	linear
(K_r, P_t)-free graphs*	linear
3-degenerate graphs*	linear
graphs of maximum degree ≤ 5 [16]	linear
graphs of maximum degree ≤ 6 [16]	polynomial
graphs of maximum average degree $< \frac{46}{11}$ [43]	polynomial
line graphs [78]	polynomial
trivially perfect graphs [79]	polynomial
threshold graphs [79]	polynomial
chordal graphs [67]	NP-complete

Table 4.1: A survey of the known results for Square Root restricted to some special graph class $\mathcal{G}$. Note that the row for planar graphs is implied by the row below. Results marked with a $*$ are results shown in this chapter.

The computational hardness of Square Root also led to a variant, which asks whether a given graph has a square root that belongs to some specified graph class $\mathcal{H}$. We denote this problem by $\mathcal{H}$-Square Root. The $\mathcal{H}$-Square Root problem is known to be polynomial-time solvable if $\mathcal{H}$ is the class of trees [73], proper interval graphs [67], bipartite graphs [66], block graphs [71], strongly chordal split graphs [72], graphs with girth at least g for any fixed $g \geq 6$ [32], ptolemaic graphs [69], 3-sun-free split graphs [69] (see [70] for an extension of the latter result to other subclasses of split graphs). In contrast, NP-completeness of this problem has been shown if $\mathcal{H}$ is the class of split graphs [67], chordal graphs [67], graphs of girth at least 4 [32] or graphs of girth at least 5 [31].

It follows from a result of Harary, Karp and Tutte [50] that every square root H of a planar square has maximum degree at most 3 and only contains blocks that are of size 4 or isomorphic to an even cycle. It follows from this that such graphs H have bounded treewidth. By "blowing up" each bag of a tree decomposition by

adding all neighbours of every vertex u to every bag that contains u, we get a tree decomposition of H^2. Hence, planar squares have bounded treewidth. As such we may use Courcelle's Theorem [21] to obtain an alternative (but comparable) proof for the polynomial-time result of Lin and Skiena [73] for Square Root restricted to planar graphs. We note that the polynomial-time algorithms for solving Square Root for graphs of maximum degree at most 6 [16] and graphs of maximum average degree less than $\frac{46}{11}$ [43] are also based on showing that the graphs which permit square roots also have bounded treewidth. Nestoridis and Thilikos [84] proved their result for minor-closed graph classes by showing boundedness of carving width. It is also possible, by using the graph minor decomposition of Robertson and Seymour [89], to show that squares of graphs from minor-closed classes have in fact bounded treewidth. Hence, it is a natural question to ask whether the technique of showing boundedness of treewidth can be used for recognising some other squares as well. In this chapter, we show that Square Root is polynomial-time solvable for graphs with clique number 3 (or equivalently, K_4-free graphs) and we prove the same result for 3-degenerate graphs and (K_r, P_t)-free graphs for every $r, t \geq 1$.

4.2 Squares of Low Clique Number

We first consider the class of K_4-free graphs. We will show that K_4-free squares have bounded treewidth. In order to do this we need the following Lemma. It was first presented by Cochefert et al. [16] however without a proof so we present a simple proof below for completeness.

Lemma 4.1 ([16])**.** *The* Square Root *problem can be solved in time* $\mathcal{O}(f(t)n)$ *for* n*-vertex graphs of treewidth at most* t*.*

Proof. By Courcelle's Theorem [21], it suffices to show that the existence of a square root can be expressed in monadic second-order logic.

Let G be an instance of Square Root. We observe that the existence of a graph H such that $G = H^2$ is equivalent to the existence of a subset $X \subseteq E_G$ such that the following properties hold:

(i) for every $uv \in E_G$, $uv \in X$ or there exists a vertex w such that $uw, wv \in X$;
(ii) for every two distinct edges $uw, vw \in X$, $uv \in E_G$;

Since both of these properties can be easily defined in monadic second-order logic, the lemma follows. □

The second lemma gives an upper bound for the treewidth of the square of a graph; it follows from the well-known fact that we can transform every tree decomposition (T, X) of a graph G into a tree decomposition of G^2 by adding, to each bag X_i of T, all the neighbours of every vertex from X_i.

Lemma 4.2. *For a graph* G*,* $\mathbf{tw}(G^2) \leq (\mathbf{tw}(G) + 1)(\Delta(G) + 1) - 1$*.*

We now show that K_4-free graphs with a square root have bounded treewidth.

Lemma 4.3. *If G is a K_4-free graph with a square root, then* $\mathbf{tw}(G) \leq 8$.

Proof. Suppose G is K_4-free and has a square root H. We observe that every vertex in H has maximum degree at most 2 since a vertex with 3 neighbours when squared would become K_4. As such it is clear that H is the disjoint union of paths and cycles and hence has treewidth at most 2. As such it follows from Lemma 4.2 that G has treewidth at most $(2+1)(2+1)-1 = 8$. □

We will make use of the following result of Bodlaender.

Lemma 4.4 ([7])**.** *For any fixed constant k, it is possible to decide in linear time whether the treewidth of a graph is at most k.*

The following result follows from Lemmas 4.1, 4.3 and 4.4.

Theorem 4.1. Square Root *can be solved in $\mathcal{O}(n)$ time for K_4-free graphs on n vertices.*

Proof. Let G be an K_4-free graph with n vertices. By Lemma 4.4 we can check in $\mathcal{O}(n)$ time whether $\mathbf{tw}(G) \leq 8$. If $\mathbf{tw}(G) > 8$, then G has no square root by Lemma 4.3. If not we solve Square Root in $\mathcal{O}(n)$ time by using Lemma 4.1. □

We now consider the class of 3-degenerate graphs where we will once again show bounded treewidth. In order to do this we need the following result.

Lemma 4.5. *Let H be a square root of a graph G. Let T be the bipartite graph with $V_T = \mathcal{C} \cup \mathcal{B}$, where partition classes $\mathcal{C}$ and $\mathcal{B}$ are the set of cut vertices and blocks of H, respectively, such that $u \in \mathcal{C}$ and $Q \in \mathcal{B}$ are adjacent if and only if Q contains u. For $u \in \mathcal{C}$, let X_u consist of u and all neighbours of u in H. For $Q \in \mathcal{B}$, let $X_Q = V_Q$. Then (T, X) is a tree decomposition of G.*

Proof. We first prove that T is a tree. For contradiction, suppose that T contains a cycle. Then this cycle is of the form $Q_1u_1 \cdots Q_pu_pQ_1$ for some integer $p \geq 2$,

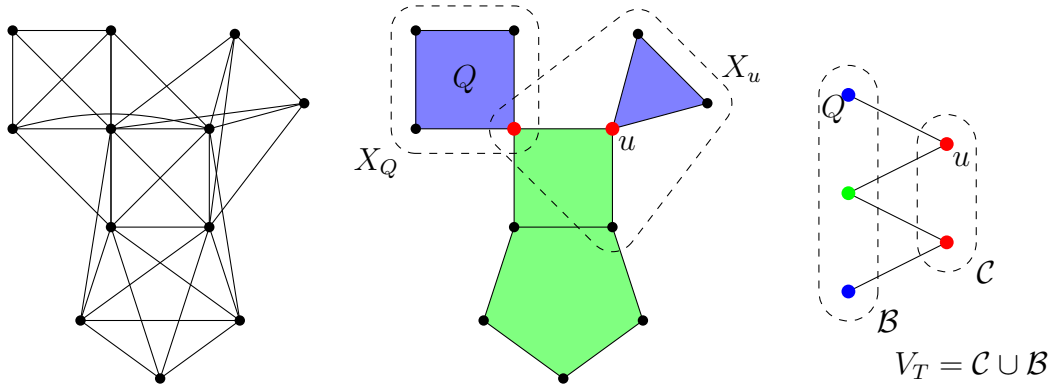


Figure 4.2: An example of a graph $G = H^2$, the graph H and the corresponding *H-tree decomposition* of G. Q is a block and u is a cut vertex.

where $Q_1, \ldots, Q_p$ are blocks of H and $u_1, \ldots, u_p$ are cut vertices of H. By using this cycle we find that H has one path from u_1 to u_2 that is contained in $H[Q_1]$ and one path from u_1 to u_2 that is contained in $H[Q_2 \cup \cdots \cup Q_p]$. This contradicts our assumption that u_1 and u_2 are cut vertices of H.

We now prove that (T, X) satisfies the three conditions (i)–(iii) of the definition of a tree decomposition. Condition (i) is satisfied, as every vertex of H, and thus every vertex of G, belongs to some block Q of H and thus to some bag X_Q. Condition (ii) is satisfied, as every two vertices x, y that are adjacent in G either belong to some common block Q of H, and thus belong to X_Q, or else have a common neighbour u in H that is a cut vertex of H, and thus belong to X_u.

In order to prove (iii), consider a vertex $x \in V_G$. First suppose that x is a cut vertex of H. Then the set of bags to which x belongs consists of bags X_Q for every block Q of H to which x belongs and bags X_u for every neighbour u of x in H that is a cut vertex of H. Note that x and any neighbour u of x in H belong to some common block of H. Hence, by definition, the corresponding nodes in T form a connected induced subtree of T (which is a star in which every edge is subdivided at most once). Now suppose that x is not a cut vertex of H. Then x is contained in exactly one block Q of H. Hence the set of bags to which x belongs consists of the bags X_Q and bags X_u for every neighbour u of x in H that is a cut vertex of H. Note that such a neighbour u belongs to Q. Hence, by definition,

the corresponding nodes in T form a connected induced subtree of T (which is a star). This completes the proof of Lemma 4.5. □

We call the tree decomposition (T, X) of Lemma 4.5 the *H-tree decomposition* of G and are now ready to prove the following lemma.

Lemma 4.6. *If G is a 3-degenerate graph with a square root, then* $\mathbf{tw}(G) \leq 3$.

Proof. Without loss of generality we assume that G is connected and has at least one edge. Let H be a square root of G. Let $\mathcal{C}$ be the set of cut vertices of H, and let $\mathcal{B}$ be the set of blocks of H. We construct the H-tree decomposition (T, X) of G (cf. Lemma 4.5). We will show that (T, X) has width at most 3.

We start with two useful observations. If $v \in V_H$, then $N_H[v]$ is a clique in G. Because G is 3-degenerate, this means that $\Delta(H) \leq 3$. For the same reason H contains no cycles of length at least 5 as a subgraph, because a square of a cycle of length at least 5 has minimum degree 4.

We claim that X_Q has size at most 4 for every $Q \in \mathcal{B}$. In order to see this, let Q be a block of H, and let $u \in V_Q$. Suppose that Q has a vertex v at distance at least 3 from u. Because Q is 2-connected, Q has two internally vertex disjoint paths that join u and v and, therefore, Q (and thus H) contains a cycle of length at least 6 which, as we saw, is not possible. We find that each vertex $v \in V_Q$ is at distance at most 2 from u. Hence, u is adjacent to all other vertices of Q in G. By the same reasoning any two vertices in Q are of distance at most 2 of each other. Hence, Q is a clique in G. As G is 3-degenerate, this means that Q is a clique in G of size at most 4. Consequently, X_Q, has size at most 4. As $\Delta(H) \leq 3$, we find that X_u has size at most 4 for every cut vertex u of H. □

Lemma 4.6, combined with Lemmas 4.4 and 4.1, leads to the following result.

Theorem 4.2. Square Root *can be solved in* $\mathcal{O}(n)$ *time for* 3*-degenerate graphs on* n *vertices.*

Proof. Let G be an 3-degenerate graph on n vertices. By Lemma 4.4 we can check in $\mathcal{O}(n)$ time whether $\mathbf{tw}(G) \leq 3$. If $\mathbf{tw}(G) > 3$, then G has no square root by Lemma 4.6. If not we solve Square Root in $\mathcal{O}(n)$ time by using Lemma 4.1. □

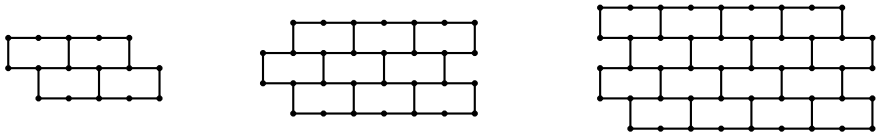


Figure 4.3: Walls of height 2, 3, and 4, respectively.

Remark. We cannot claim any upper bound for the treewidth of 4-degenerate graphs with a square root. In order to see this, take a wall (see Figure 4.3) and subdivide each edge three times, that is, replace each edge uv by a path $uabcv$ where a, b, c are three new vertices. This gives us a graph H, such that H^2 is 4-degenerate. In order to see the latter, note that every "b-type" vertex has degree 4 in H^2 and that after removing all degree-4 vertices, we obtain a disjoint number of copies of K_4, each of which is 4-degenerate. A wall of height h has treewidth $\Omega(h)$ (see, for example, [26]). As subdividing an edge and adding edges does not decrease the treewidth of a graph, this means that the graph H^2 can have arbitrarily large treewidth.

We now consider (K_r, P_t)-free graphs (that is, graphs with no long induced path P_t and no large complete subgraph K_r). We let $K_{s,s}$ denote the complete bipartite graph in which both partition classes have s vertices. We need a result of Atminas, Lozin and Razgon.

Lemma 4.7 ([3])**.** *For any two integers s and t, there exists an integer $b(s,t)$ such that any graph of treewidth at least $b(s,t)$ contains the path P_t as an induced subgraph or the complete bipartite graph $K_{s,s}$ as a (not necessarily induced) subgraph.*

Lemma 4.7 enables us to prove the following lemma.

Lemma 4.8. *For every two fixed integers $r,t \geq 1$, if G is a (K_r, P_t)-free graphs with a square root, then $\mathbf{tw}(G) \leq b((r-2)^2+1,t)$.*

Proof. Let $r,t \geq 1$. For contradiction, assume that the class of (K_r, P_t)-free graphs with a square root has unbounded treewidth. Then there exists a (K_r, P_t)-free graph G with a square root such that G has treewidth at least $b(s,t)$, where $b(s,t)$ is the constant in Lemma 4.7 for a sufficiently large integer s. Then, by Lemma 4.7, we find that G contains a subgraph F isomorphic to $K_{s,s}$. We denote the vertex sets of the two partition classes of F as A and B. Let H be a square root of G. We observe that there must be s paths of length at most 2 in H between a vertex $u \in A$ and the s vertices of B. Let H' be the subgraph induced by these paths in H. A vertex of degree d in H would result in a clique of size $d+1$ in G. Since G is K_r-free we see that each vertex in H and hence in H' has degree at most $r-2$. As such we see that there are at most $r-2$ vertices adjacent to to u in H' and at most $r-2$ vertices (including u) adjacent to each of those. Hence $|H'| \leq (r-2)^2+1$. Since we know that H' includes at least u and its s neighbours B in G we have a contradiction when we choose $s > (r-2)^2$. Hence we see that any (K_r, P_t)-free graph with treewidth at least $b((r-2)^2+1,t)$ does not have a square root. □

Using the same reasoning as in Theorems 4.1 and 4.2, we find that Lemma 4.8, combined with Lemmas 4.4 and 4.1, leads to the following result.

Theorem 4.3. *For every two integers $r,t \geq 1$,* Square Root *can be solved in time $\mathcal{O}(n)$ for (K_r, P_t)-free graphs on n vertices.*

4.3 Conclusions

The motivation for this research stems from the fact that the hardness reductions for Square Root rely on the creation of large cliques [67]. This leads us to that natural question as to whether Square Root is tractable on graphs with bounded clique number. As such we pose the following open question: Does there exist an integer r such that Square Root is hard when restricted to the class of K_r-free graphs? It may also be the case the Square Root is tractable when restricted to the class of graphs with bounded clique number. In this chapter we have shown that it is linear time solvable when restricted to the the classes of K_4-free graphs and (K_r, P_t)-free graphs. We believe that the logical next step would be in determining the complexity of Square Root when restricted to K_5-free graphs.

In addition we showed that Square Root is linear time solvable when restricted to 3-degenerate graphs, however, as mentioned this method would not work for 4-degenerate graphs as we cannot claim any upper bound for the treewidth of such graphs.

We recall the summary of the known results for Square Root in Table 4.1 on page 79. As can be seen from the table, the computational complexity of Square Root is unknown for several other well-known graph classes. In particular, we recall the open problems of Milanic and Schaudt [79], who asked about the complexity of Square Root restricted to split graphs and cographs.

Chapter 5

Surjective H-Colouring with 2-reflexive H

A homomorphism from a graph G to a graph H is a vertex mapping f from the vertex set of G to the vertex set of H such that there is an edge between vertices $f(u)$ and $f(v)$ of H whenever there is an edge between vertices u and v of G.

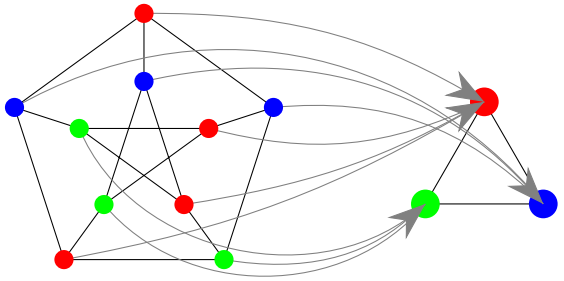


Figure 5.1: An example of a homomorphism from the 3-colourable Peterson graph to the irreflexive K_3.

The H-Colouring problem is to decide if a graph G allows a homomorphism to a fixed graph H. The H-Colouring problem generalises the well known k-Colouring since a graph is k-colourable if and only if it is K_k-colourable. We continue a study on a variant of this problem, the Surjective H-Colouring problem, which requires the homomorphism to be vertex-surjective. We show that this problem is NP-complete for every connected graph H that has exactly two vertices with a self-loop as long as these two vertices are not adjacent. This result enables us to classify the complexity of Surjective H-Colouring for every fixed graph H on at most four vertices.

This result has been submitted to Computability in Europe 2017: Unveiling Dynamics and Complexity (CiE 2017), Turku, Finland and can be found on arXiv [41]. A journal paper is currently being written.

5.1 Introduction

The well-known COLOURING problem is to decide if the vertices of a given graph can be properly coloured with at most k colours for some given integer k. If we exclude k from the input and assume it is fixed, we obtain the k-COLOURING problem. A *homomorphism* from a graph $G = (V_G, E_G)$ to a graph $H = (V_H, E_H)$ is a vertex mapping $f : V_G \to V_H$, such that there is an edge between $f(u)$ and $f(v)$ in H whenever there is an edge between u and v in G. We observe that k-COLOURING is equivalent to the problem of asking if a graph allows a homomorphism to the complete graph K_k on k vertices. Hence, a natural generalisation of the k-COLOURING problem is the H-COLOURING problem, which asks if a given graph allows a homomorphism to an arbitrary fixed graph H. We call this fixed graph H the *target graph*. Throughout the paper we consider undirected graphs with no multiple edges. We assume that an input graph G contains no vertices with self-loops (we call such vertices *reflexive*), whereas a target graph H may contain such vertices. We call H *reflexive* if all its vertices are reflexive, and *irreflexive* if all its vertices are irreflexive.

For a survey on graph homomorphisms we refer the reader to the textbook of Hell and Nešetřil [53]. Here, we will discuss the H-COLOURING problem, a number of its variants and their relations to each other. In particular, we will focus on the *surjective* variant: a homomorphism f from a graph G to a graph H is *(vertex-)surjective* if f is surjective, that is, if for every vertex $x \in V_H$ there exists at least one vertex $u \in V_G$ with $f(u) = x$.

The computational complexity of H-COLOURING has been determined completely. The problem is trivial if H contains a reflexive vertex u (we can map each vertex of the input graph to u). If H has no reflexive vertices, then the Hell-Nešetřil dichotomy theorem [52] tells us that H-Colouring is solvable in polynomial time if H is bipartite and that it is NP-complete otherwise.

The List H-Colouring problem takes as input a graph G and a function L that assigns to each $u \in V_G$ a list $L(u) \subseteq V_H$. The question is whether G allows a homomorphism f to the target H with $f(u) \in L(u)$ for every $u \in V_G$. Feder, Hell and Huang [33] proved that List H-Colouring is polynomial-time solvable if H is a bi-arc graph and NP-complete otherwise (we refer to [33] for the definition of a bi-arc graph). A homomorphism f from G to an induced subgraph H of G is a *retraction* if $f(x) = x$ for every $x \in V_H$, and we say that G *retracts to* H. A retraction from G to H can be viewed as a list-homomorphism: choose $L(u) = \{u\}$ if $u \in V_H$, and $L(u) = V_H$ if $u \in V_G \setminus V_H$. The corresponding decision problem is called H-Retraction. The computational complexity of H-Retraction has not yet been classified. Feder et al. [34] determined the complexity of the H-Retraction problem whenever H is a pseudo-forest (a graph in which every connected component has at most one cycle). They also showed that H-Retraction is NP-complete if H contains a connected component in which the reflexive vertices induce a disconnected graph.

As mentioned, we impose a (vertex-)surjectivity condition on the graph homomorphism. Such a condition can be imposed locally or globally. If we require a homomorphism f from a graph G to a graph H to be surjective when restricted to the open neighbourhood of every vertex u of G, we say that f is an *H-role assignment*. The corresponding decision problem is called H-Role Assignment and its computational complexity has been fully classified [37]. We refer to the survey of Fiala and Kratochvíl [36] for further details on locally constrained homomorphisms and from here on only consider global surjectivity.

It has been shown that deciding whether a given graph G allows a surjective homomorphism to a given graph H is NP-complete even if G and H both belong to one of the following graph classes: disjoint unions of paths; disjoint unions of complete graphs; trees; connected cographs; connected proper interval graphs; and connected split graphs [45]. Hence it is natural, just as before, to fix H which yields the following problem:

Surjective H-Colouring	
Instance:	a graph G.
Question:	does there exist a surjective homomorphism from G to H?

We emphasise that we are considering vertex-surjectivity and that being vertex-surjective is a different condition than being edge-surjective. A homomorphism from a graph G to a graph H is called *edge-surjective* or a *compaction* if for any edge $xy \in E_H$ with $x \neq y$ there exists an edge $uv \in E_G$ with $f(u) = x$ and $f(v) = y$. Note that the edge-surjectivity condition does not hold for any self-loops $xx \in E_H$. If f is a compaction from G to H, we say that G *compacts* to H. The corresponding decision problem is known as the H-Compaction problem. A full classification of this problem is still wide open. However partial results are known, for example when H is a reflexive cycle, an irreflexive cycle, or a graph on at most four vertices [94–96], or when G is restricted to some special graph class [97]. Vikas also showed that whenever H-Retraction is polynomial-time solvable, then so is H-Compaction [95]. Whether the reverse implication holds is not known. A complete complexity classification of Surjective H-Colouring is also still open. Below we survey the known results.

We first consider irreflexive target graphs H. The Surjective H-Colouring problem is NP-complete for every such graph H if H is non-bipartite, as observed by Golovach et al. [46]. The straightforward reduction is from the corresponding H-Colouring problem, which is NP-complete due to the aforementioned Hell-Nešetřil dichotomy theorem. However, the complexity classifications of H-Colouring and Surjective H-Colouring do not coincide: there exist bipartite graphs H for which Surjective H-Colouring is NP-complete, for instance when H is the graph obtained from a 6-vertex cycle to each of which vertices we add a path of length 3 [6], or when H is the 6-vertex cycle itself [**?**].

We now consider target graphs with at least one reflexive vertex. Unlike the H-Colouring problem, the presence of a reflexive vertex does not make the

Surjective H-Colouring problem trivial to solve. We call a connected graph *loop-connected* if all its reflexive vertices induce a connected subgraph. Golovach, Paulusma and Song [46] showed that if H is a tree (in this context, a connected graph with no cycles of length at least 3) then Surjective H-Colouring is polynomial-time solvable if H is loop-connected and NP-complete otherwise. As such the following question is natural:

Is Surjective H-Colouring NP*-complete for every connected graph* H *that is not loop-connected?*

The reverse statement is not true (if P$\neq$ NP): Surjective H-Colouring is NP-complete when H is the 4-vertex cycle C_4^* with a self-loop in each of its vertices. This result has been shown by Martin and Paulusma [76] and independently by Vikas, as announced in [97]. Recall also that Surjective H-Colouring is NP-complete if H is irreflexive (and thus loop-connected) and non-bipartite.

It is known that Surjective H-Colouring is polynomial-time solvable whenever H-Compaction is [6]. Recall that H-Compaction is polynomial-time solvable whenever H-Retraction is [95]. Hence, for instance, the aforementioned result of Feder, Hell and Huang [33] implies that Surjective H-Colouring is polynomial-time solvable if H is a bi-arc graph. We also recall that H-Retraction is NP-complete whenever H is a connected graph that is not loop-connected [34]. Hence, an affirmative answer to the above question would mean that for these target graphs H the complexities of H-Retraction, H-Compaction and Surjective H-Colouring coincide.

In Figure 5.2 we display the relationships between the different problems discussed. In particular, it is a major open problem whether the computational complexities of H-Compaction, H-Retraction and Surjective H-Colouring coincide for each target graph H. Even showing this for specific cases, such as the case $H = C_4^*$, has been proven to be non-trivial. If it is true, it would relate the Surjective H-Colouring problem to a well-known conjecture of Feder

and Vardi [35], which states that the $\mathcal{H}$-Constraint Satisfaction problem has a dichotomy when $\mathcal{H}$ is some fixed finite target structure and which is equivalent to conjecturing that H-Retraction has a dichotomy [35]. We refer to the survey of Bodirsky, Kara and Martin [6] for more details on the Surjective H-Colouring problem from a constraint satisfaction point of view.

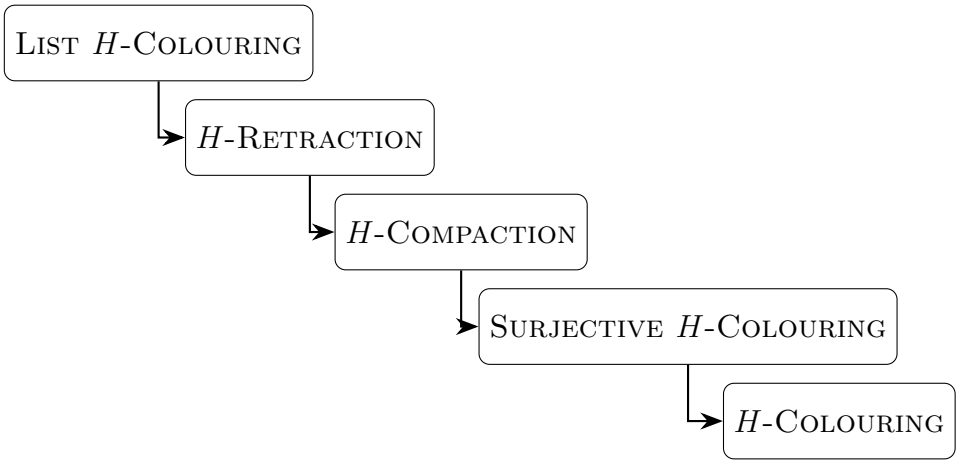


Figure 5.2: Relations between Surjective H-Colouring and its variants. An arrow from one problem to another indicates that the latter problem is polynomial-time solvable for a target graph H if the former is polynomial-time solvable for H. Reverse arrows do not hold for the leftmost and rightmost arrows, as witnessed by the reflexive 4-vertex cycle for the rightmost arrow and by any reflexive tree that is not a reflexive interval graph for the leftmost arrow (Feder, Hell and Huang [33] showed that the only reflexive bi-arc graphs are reflexive interval graphs). It is not known if the reverse direction holds for the two middle arrows.

5.1.1 Our Results

We present further progress on the research question of whether Surjective H-Colouring is NP-complete for every connected graph H that is not loop-connected. We first consider the case where the target graph H is a connected graph with exactly two reflexive vertices that are non-adjacent. In Section 5.2 we prove that Surjective H-Colouring is indeed NP-complete for every such target graph H. In the same section we slightly generalise this result by showing that it holds even if the reflexive vertices of H can be partitioned into two non-adjacent sets of twin vertices. This enables us to classify in Section 5.3 the

computational complexity of Surjective H-Colouring for every graph H on at most four vertices, just as Vikas [96] did for the H-Compaction problem.

5.2 Two Non-Adjacent Reflexive Vertices

We say that a graph is 2-*reflexive* if it contains exactly 2 reflexive vertices *that are non-adjacent*. In this section we will prove that Surjective H-Colouring is NP-complete whenever H is connected and 2-reflexive. The problem is readily seen to be in NP. Our NP-hardness reduction uses similar ingredients as the reduction of Golovach, Paulusma and Song [46] for proving NP-hardness when H is a tree that is not loop-connected. There are, however, a number of differences. For instance, we will reduce from a factor cut problem instead of the less general matching cut problem used in [46]. We will explain these two problems and prove NP-hardness for the former one in Section 5.2.1. Then in Section 5.2.2 we give our hardness reduction.

5.2.1 Factor Cuts

Let $G = (V_G, E_G)$ be a connected graph. For $v \in V_G$ and $E \subseteq E_G$, let $d_E(v)$ denote the number of edges of E incident with v. For a partition (V_1, V_2) of V_G, let $E_G(V_1, V_2)$ denote the set of edges between V_1 and V_2 in G.

Let i and j be positive integers, $i \leq j$. Let (V_1, V_2) be a partition of V_G and let $M = E_G(V_1, V_2)$. Then (V_1, V_2) is an (i, j)-*factor cut* of G if, for all $v \in V_1$, $d_M(v) \leq i$, and, for all $v \in V_2$, $d_M(v) \leq j$. Observe that if a vertex v exists with degree at most j, then there is a trivial (i, j)-factor cut $(V \setminus \{v\}, \{v\})$. Two distinct vertices s and t in V_G are (i, j)-*factor roots* of G if, for each (i, j)-factor cut (V_1, V_2) of G, s and t belong to different parts of the partition and, if $i < j$, $s \in V_1$ and $t \in V_2$ (of course, if $i = j$, we do not require the latter condition as (V_2, V_1) is also an (i, j)-factor cut). We note that when no (i, j)-factor cut exists, every pair of vertices is a pair of (i, j)-factor roots. We define the following decision problem.

(i,j)-Factor Cut with Roots	
Instance:	a connected graph G with roots s and t.
Question:	does G have an (i,j)-factor cut?

We emphasise that the (i,j)-factor roots are given as part of the input. That is, the problem asks whether or not an (i,j)-factor cut (V_1,V_2) exists, but we know already that if it does, then s and t belong to different parts of the partition. That is, we actually define (i,j)-Factor Cut with Roots to be a promise problem in which we assume that if an (i,j)-factor cut exists then it has the property that s and t belong to different parts of the partition. The promise class may not itself be polynomially recognisable but one may readily find a subclass of it that is polynomially recognisable and includes all the instances we need for NP-hardness. In fact this will become clear when reading our proof but we refer also to [46] where such a subclass is given for the case $(i,j)=(1,1)$.

A $(1,1)$-factor cut (V_1,V_2) of G is also known as a *matching cut* as no two edges in $E_G(V_1,V_2)$ have a common end-vertex, that is, $E_G(V_1,V_2)$ is a *matching*. Similarly $(1,1)$-Factor Cut with Roots is known as Matching Cut with Roots and was proved NP-complete by Golovach, Paulusma and Song [46] (by making an observation about the proof of the result of Patrignani and Pizzonia [85] that deciding if any given graph has a matching cut is NP-complete).

We will prove the NP-completeness of (i,j)-Factor Cut with Roots after first presenting a helpful lemma (a *clique* is a subset of vertices of G that are pairwise adjacent to each other).

Lemma 5.1. *Let i, j and k be positive integers where $i \leq j$ and $k > i+j$. Let G be a graph that contains a clique K on k vertices. Then, for every (i,j)-factor cut (V_1,V_2) of G, either $V_K \subseteq V_1$ or $V_K \subseteq V_2$.*

Proof. If the lemma is false, then for some (i,j)-factor cut (V_1,V_2), we can choose $v_1 \in V_1 \cap V_K$ and $v_2 \in V_2 \cap V_K$. Let $M = E_G(V_1,V_2)$. Since every vertex in $V_1 \cap V_K$

is linked by an edge of M to v_2 and every vertex in $V_2 \cap V_K$ is linked by an edge of M to v_1, we have $d_M(v_1) + d_M(v_2) \geq k > i + j$, contradicting the definition of an (i, j)-factor cut. □

Theorem 5.1. *Let i and j be positive integers, $i \leq j$. Then* (i, j)-Factor Cut with Roots *is* NP*-complete.*

Proof. If $i = j = 1$, then the problem is Matching Cut with Roots which, as we noted, is known to be NP-complete [46]. We split the remaining cases in two according to whether or not $i = 1$. In each case, we construct a polynomial time reduction from Matching Cut with Roots. In particular, we take an instance (G, s, t) of Matching Cut with Roots, and construct a graph G' that is a supergraph of $G = (V, E)$ and show that

1. (G', s, t) is an instance of (i, j)-Factor Cut with Roots (that is, if G' has an (i, j)-factor cut (V_1', V_2'), then $s \in V_1$ and $t \in V_2$ or, possibly, vice versa if $i = j$),
2. if G' has an (i, j)-factor cut, then G has a matching cut, and
3. if G has a matching cut, then G' has an (i, j)-factor cut.

We note that (1) is an atypical feature of an NP-completeness proof as, unusually for (i, j)-Factor Cut with Roots, it is not immediate to recognise a problem instance. We let $n = |V|$.

Case 1: $i = 1$.

Let $k = \max\{(n - 1)(j - 1), 1 + j\}$. Construct G' from G by first adding a complete graph K on k vertices and adding edges from s to every vertex of V_K. Then, for each $v \in V_G \setminus \{s\}$, add edges from v to $j - 1$ vertices of K in such a way that no vertex of V_K has more than one neighbour in $V_G \setminus \{s\}$.

Let (V_1', V_2') be a $(1, j)$-factor cut of G'. The vertices of $\{s\} \cup V_K$ induce a clique on $1 + k > 1 + j$ vertices. So, by Lemma 5.1, $\{s\} \cup V_K \subseteq V_1'$ or $\{s\} \cup V_K \subseteq V_2'$.

Suppose that $\{s\} \cup V_K \subseteq V_2'$. Then V_G must contain vertices of both V_1' (otherwise V_1' would be empty) and V_2' (at least s). Thus, as G is connected, we can find a vertex $v \in V_1' \cap V_G$ that has a neighbour in $V_2' \cap V_G$. But v also has $j - 1 \geq 1$ neighbours in V_K and so has at least 2 neighbours in V_2', contradicting the definition of a $(1, j)$-factor cut.

So we must have that $\{s\} \cup V_K \subseteq V_1'$. Let $V_1 = V_1' \cap V_G$ and $V_2 = V_2'$ be a partition of V_G, and let $M = E_G(V_1, V_2)$ and $M' = E_G(V_1', V_2')$ and notice that M' is the union of M and, for each $v \in V_2$, the $j - 1$ edges from v to V_K. For each $v \in V_1$, $d_M(v) = d_{M'}(v) \leq 1$. For each $v \in V_2$, $d_M(v) = d_{M'}(v) - (j - 1) \leq 1$. So (V_1, V_2) is a matching cut of G; this proves (2). As $s \in V_1$, we have, by the definition of factor roots, $t \in V_2$; this proves (1).

To prove (3), we note that if (V_1, V_2) is a matching cut of G, then we can assume that $s \in V_1$ and $t \in V_2$ (else relabel them for the purpose of constructing G'), and then $(V_1 \cup V_K, V_2)$ is a $(1, j)$-factor cut of G'.

Case 2: $i \geq 2$.

Let $k = \max\{(n-1)(j-1), i+j\}$. Construct G' from G by first adding a complete graph K^s on k vertices and adding edges from s to every vertex of V_{K^s}, and then adding a complete graph K^t on k vertices and adding edges from t to every vertex of V_{K^t}. Then, for each $v \in V_G \setminus \{s\}$, add edges from v to $j - 1$ vertices of K^s in such a way that no vertex of V_{K^s} has more than one neighbour in $V_G \setminus \{s\}$. Afterwards, for each $v \in V_G \setminus \{t\}$, add edges from v to $i - 1$ vertices of K^t in such a way that no vertex of V_{K^t} has more than one neighbour in $V_G \setminus \{t\}$.

Let (V_1', V_2') be an (i, j)-factor cut of G'. The vertices of $\{s\} \cup V_{K^s}$ induce a clique on at least $1 + k > i + j$ vertices. So, by Lemma 5.1, $\{s\} \cup V_{K^s} \subseteq V_1'$ or $\{s\} \cup V_{K^s} \subseteq V_2'$. Similarly $\{t\} \cup V_{K^t} \subseteq V_1'$ or $\{t\} \cup V_{K^t} \subseteq V_2'$.

Suppose that $\{s\} \cup V_{K^s}$ and $\{t\} \cup V_{K^t}$ are both subsets of V_1'. Then V_G must contain vertices of both V_1' (at least s and t) and V_2' (else it would be empty). Thus, as G is connected, we can find a vertex $v \in V_2' \cap V_G$ that has a neighbour in $V_1' \cap V_G$. But v also has $j-1$ neighbours in V_{K^s} and $i-1$ neighbours in V_{K^t} and so has at least $1+(i-1)+(j-1) = i+j-1 > j \geq i$ neighbours in V_2', contradicting the definition of an (i,j)-factor. By an analogous argument $\{s\} \cup V_{K^s}$ and $\{t\} \cup V_{K^t}$ cannot both be subsets of V_2'.

Suppose that $i < j$ and $\{s\} \cup V_{K^s} \subseteq V_2'$. As G is connected and V_G contains vertices of both V_1' and V_2', we can find a vertex $v \in V_1' \cap V_G$ that has a neighbour in $V_2' \cap V_G$. But v also has $j-1 > i-1$ neighbours in V_{K^s} and so has more than i neighbours in V_2', contradicting the definition of a (i,j)-factor.

Thus we have that $\{s\} \cup V_{K^s}$ and $\{t\} \cup V_{K^t}$ are subsets of separate parts and, moreover, either $\{s\} \cup V_{K^s} \subseteq V_1'$ or $i = j$. Thus (1) is proved, and we have, in either case, that each vertex in $V_1' \cap V_G$ is joined by $i-1$ edges to vertices in $V_2' \setminus V_G$, and each vertex in $V_2' \cap V_G$ is joined by $j-1$ edges to vertices in $V_1' \setminus V_G$. Therefore each vertex in $V_1' \cap V_G$ is joined to at most one vertex in $V_2' \cap V_G$, and each vertex in $V_2' \cap V_G$ is joined to at most one vertex in $V_1' \cap V_G$. Thus $(V_1' \cap V_G, V_2' \cap V_G)$ is a matching cut of G. This proves (2).

To prove (3), we note that if (V_1, V_2) is a matching cut of G, then we can assume that $s \in V_1$ and $t \in V_2$ (else relabel them for the purpose of constructing G'), and then $(V_1 \cup V_{K^s}, V_2 \cup V_{K^t})$ is an (i,j)-factor cut of G'. □

5.2.2 The Hardness Reduction

Let H be a connected 2-reflexive target graph. Let p and q be the two (non-adjacent) reflexive vertices of H. The *length* of a path is its number of edges. The *distance* between two vertices u and v in a graph G is the length of a shortest path between them and is denoted $\mathrm{dist}_G(u,v)$. We define two induced

subgraphs H_1 and H_2 of H whose vertex sets partition V_H. First H_1 contains those vertices of H that are closer to p than to q; and H_2 contains those vertices that are at least as close to q as to p (so contains any vertex equidistant to p and q). That is, $V_{H_1} = \{v \in V_H : \text{dist}_H(v,p) < \text{dist}_H(v,q)\}$ and $V_{H_2} = \{v \in V_H : \text{dist}_H(v,q) \leq \text{dist}_H(v,p)\}$. See Figure 5.3 for an example.

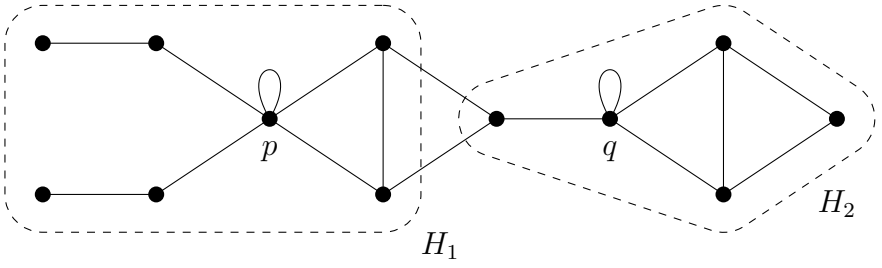


FIGURE 5.3: An example of the construction of graphs H_1 and H_2 from a connected 2-reflexive target graph H with $\omega = 3$.

The following lemma follows immediately from our assumption that H is connected.

Lemma 5.2. *Both H_1 and H_2 are connected. Moreover,* $\text{dist}_{H_1}(x,p) = \text{dist}_H(x,p)$ *for every $x \in V_{H_1}$ and* $\text{dist}_{H_2}(x,q) = \text{dist}_H(x,q)$ *for every $x \in V_{H_2}$.*

Let ω denote the size of a largest clique in H. From graphs H_1 and H_2 we construct graphs F_1 and F_2, respectively, in the following way:

1. for each $x \notin \{p,q\}$, create a vertex t_x^1;
2. for p, create ω vertices $t_p^1, \ldots, t_p^\omega$;
3. for q, create ω vertices $t_q^1, \ldots, t_q^\omega$;
4. for $i = 1, 2$, add an edge in F_i between any two vertices t_x^h and t_y^j if and only if xy is an edge of E_{H_i}.

We note that F_1 is the graph obtained by taking H_1 and replacing p by a clique of size ω. Similarly, F_2 is the graph obtained by taking H_2 and replacing q by a

clique of size ω. We say that $t_p^1, \ldots, t_p^\omega$ are the *roots* of F_1 and that $t_q^1, \ldots, t_q^\omega$ are the *roots* of F_2. Figure 5.4 shows an example of the graphs F_1 and F_2 obtained from the graph H in Figure 5.3.

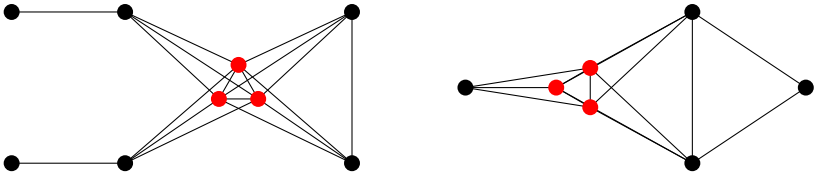


Figure 5.4: The graphs F_1 (left) and F_2 (right) resulting from the graph H in Figure 5.3.

Let $\ell = \text{dist}_H(p, q) \geq 2$ denote the distance between p and q. Let N_p be the set of neighbours of p that are each on some shortest path (thus of length ℓ) from p to q in H. Let r_p be the size of a largest clique in N_p. We define N_q and r_q similarly. We will reduce from (r_p, r_q)-Factor Cut with Roots, which is NP-complete due to Theorem 5.1. Hence, consider an instance (G, s, t) of (r_p, r_q)-Factor Cut with Roots, where G is a connected graph and s and t form the (ordered) pair of (r_p, r_q)-factor roots of G. Recall that we assume that G is irreflexive.

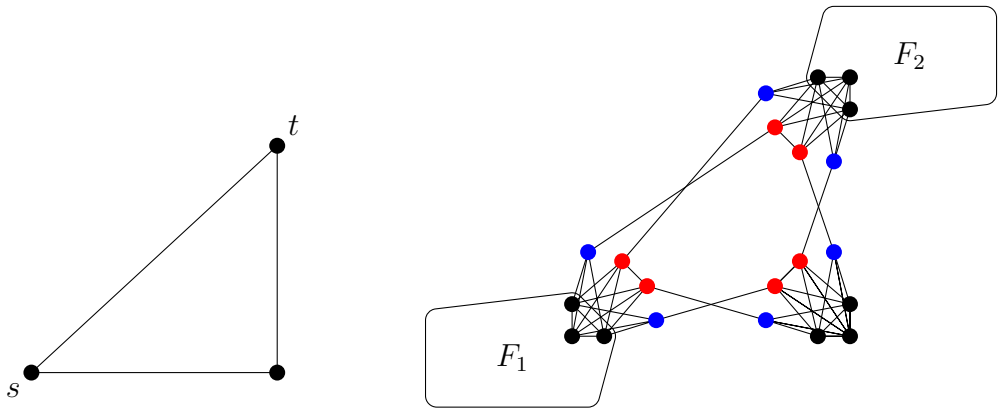
We say that we *identify* two vertices u and v of a graph when we remove them from the graph and replace them with a single vertex that we make adjacent to every vertex that was adjacent to u or v. Our aim is to create a graph based on G where each vertex is replaced by cliques so large that in any homomorphism to H, they must be mapped to reflexive vertices. We then construct paths between these cliques that must map to shortest paths between the reflexive vertices such that vertices from the same component of an (i, j)-factor cut must be mapped to the same reflexive vertex. From F_1, F_2, and G we construct a new graph G' as follows:

1. For each edge $e = uv \in E_G$, we do as follows. We create four vertices, $g_{u,e}^{\text{r}}$, $g_{u,e}^{\text{b}}$, $g_{v,e}^{\text{r}}$ and $g_{v,e}^{\text{b}}$. We also create two paths P_e^1 and P_e^2, each of length $\ell - 2$, between $g_{u,e}^{\text{r}}$ and $g_{v,e}^{\text{b}}$, and between $g_{v,e}^{\text{r}}$ and $g_{u,e}^{\text{b}}$, respectively. If $\ell = 2$ we identify $g_{u,e}^{\text{r}}$ and $g_{v,e}^{\text{b}}$ and $g_{v,e}^{\text{r}}$ and $g_{u,e}^{\text{b}}$ to get paths of length 0.

2. For each vertex $u \in V_G$, we do as follows. First we construct a clique C_u on ω vertices. We denote these vertices by $g_u^1, \ldots, g_u^\omega$. We then make every vertex in C_u adjacent to both $g_{u,e}^{\mathrm{r}}$ and $g_{u,e}^{\mathrm{b}}$ for every edge e incident to u; we call $g_{u,e}^{\mathrm{r}}$ and $g_{u,e}^{\mathrm{b}}$ a *red* and *blue* neighbour of C_u, respectively; if $\ell = 2$, then the vertex obtained by identifying two vertices $g_{u,e}^{\mathrm{r}}$ and $g_{v,e}^{\mathrm{b}}$, or $g_{v,e}^{\mathrm{r}}$ and $g_{u,e}^{\mathrm{b}}$ is simultaneously a red neighbour of one clique and a blue neighbour of another one. Finally, for every two edges e and e' incident to u, we make $g_{u,e}^{\mathrm{r}}$ and $g_{u,e'}^{\mathrm{r}}$ adjacent, that is, the set of red neighbours of C_u form a clique, whereas the set of blue neighbours form an independent set.

3. We add F_1 by identifying t_p^i and g_s^i for $i = 1, \ldots, \omega$, and we add F_2 by identifying t_q^i and g_t^i for $i = 1, \ldots, \omega$. We denote the vertices in F_1 and F_2 in G' by their label t_x^i in F_1 or F_2.

See Figure 5.5 for an example of a graph G'.



(A) An example of a graph G with a $(1,2)$-factor cut with $(1,2)$-factor roots s and t.

(B) The corresponding graph G' where H is a 2-reflexive target graph with $\ell = 3$ and $\omega = 3$.

Figure 5.5: An example of a graph G and the corresponding graph G'.

The next lemma describes a straightforward property of graph homomorphisms that will prove useful.

Lemma 5.3. *If there exists a homomorphism $h : G' \to H$ then* $\mathrm{dist}_{G'}(u,v) \geq \mathrm{dist}_H(h(u), h(v))$ *for every pair of vertices $u, v \in V_{G'}$.*

We now prove the key property of our construction.

Lemma 5.4. *For every homomorphism h from G' to H, there exists at least one clique C_a with $p \in h(C_a)$ and at least one clique C_b with $q \in h(C_b)$.*

Proof. Since for each $u \in V_G$ and any edge e incident to u, every clique $C_u \cup \{g^{\mathrm{r}}_{u,e}\}$ in G' is of size at least $\omega + 1$, we find that h must map at least two of its vertices to a reflexive vertex, so either to p or q. Hence, for every $u \in V_G$, we find that h maps at least one vertex of C_u to either p or q.

We prove the lemma by contradiction. We will assume that h does not map any vertex of any C_u to q, thus $p \in h(C_u)$ for all $u \in V_G$. We will note later that if instead $q \in h(C_u)$ for all $u \in V_G$ we can obtain a contradiction in the same way.

We consider two vertices $t^i_p \in F_1$ and $t^j_q \in F_2$ such that $h(t^i_p) = h(t^j_q) = p$. Without loss of generality let $i = j = 1$. We shall refer to these vertices as t_p and t_q respectively. We now consider a vertex $v \in V_{F_1} \cup V_{F_2}$. By Lemma 5.3, $\mathrm{dist}_{G'}(v, t_p) \geq \mathrm{dist}_H(h(v), p)$ and $\mathrm{dist}_{G'}(v, t_q) \geq \mathrm{dist}_H(h(v), p)$. In other words:

$$\min\left(\mathrm{dist}_{G'}(v, t_p), \mathrm{dist}_{G'}(v, t_q)\right) \geq \mathrm{dist}_H(h(v), p).$$

In fact by applying Lemma 5.3 we can generalise this further to any vertex mapped to p by h:

$$\min_{w \in h^{-1}(p)} \left(\mathrm{dist}_{G'}(v, w)\right) \geq \mathrm{dist}_H(h(v), p). \tag{5.1}$$

For every $v \in V_{G'}$ we define a value $\mathcal{D}(v)$ as follows:

$$\mathcal{D}(v) = \begin{cases} \mathrm{dist}_{F_1}(v, t_p) & \text{if } v \in F_1 \\ \mathrm{dist}_{F_2}(v, t_q) & \text{if } v \in F_2 \\ \lfloor \ell/2 \rfloor & \text{otherwise} \end{cases}$$

Claim 5.1. $\mathcal{D}(v) \geq \min_{w \in h^{-1}(p)} \left(\mathrm{dist}_{G'}(v, w)\right) \geq \mathrm{dist}_H(h(v), p)$ *for all $v \in V_{G'}$.*

We prove Claim 5.1 by showing that $\mathcal{D}(v) \geq \min_{w \in h^{-1}(p)} (\text{dist}_{G'}(v, w))$, which suffices due to (5.1). First suppose $v \in V_{F_1} \cup V_{F_2}$. We may assume, without loss of generality, that $v \in V_{F_2}$. So $\mathcal{D}(v) = \text{dist}_{F_2}(v, t_q) = \text{dist}_{G'}(v, t_q) \geq \min_{w \in h^{-1}(p)} (\text{dist}_{G'}(v, w))$, as $t_q \in h^{-1}(p)$.

Now suppose $v \notin V_{F_1} \cup V_{F_2}$. Then v either belongs to a clique C_u or is a vertex of a path P_e^1 or P_e^2 between two cliques. If v belongs to a clique or is an end-vertex of such a path, then v is either in $h^{-1}(p)$ or adjacent to a vertex in $h^{-1}(p)$ (since at least one vertex in C_u maps to p). Hence $\mathcal{D}(v) = \lfloor \ell/2 \rfloor \geq 1 \geq \min_{w \in h^{-1}(p)} (\text{dist}_{G'}(v, w))$. Finally, suppose v is an inner vertex of a path P_e^1 or P_e^2. By definition, such a path has length $\ell - 2$. Then v is at most distance $\lfloor (\ell - 2)/2 \rfloor$ from a vertex in a clique, which we know is either in $h^{-1}(p)$ or adjacent to a vertex in $h^{-1}(p)$. Hence $\mathcal{D}(v) = \lfloor \ell/2 \rfloor = \lfloor (\ell - 2)/2 \rfloor + 1 \geq \min_{w \in h^{-1}(p)} (\text{dist}_{G'}(v, w))$. This proves Claim 5.1.

Claim 5.2. *If there exists a surjective homomorphism from G' to H, then for any integer $d \geq \ell$:*

$$\left| \left\{ t_w^1 \in V_{F_1} \cup V_{F_2} : \mathcal{D}(t_w^1) \geq d \right\} \right| \geq \left| \{ w \in V_H : \text{dist}_H(w, p) \geq d \} \right|.$$

We prove Claim 5.2 as follows. Using the fact that with a surjective homomorphism every vertex must be mapped to, we see from Lemma 5.3 that if there are n vertices in H which are at a distance d from p, there must be at least n vertices in G' that are at distance at least d from every vertex that maps to p. This means we can say for any distance $d \geq 0$:

$$\left| \{ v \in V_{G'} : \min_{w \in h^{-1}(p)} (\text{dist}_{G'}(v, w)) \geq d \} \right| \geq \left| \{ w \in V_H : \text{dist}_H(w, p) \geq d \} \right|.$$

Combining this inequality with Claim 5.1 yields, for every distance $d \geq 0$:

$$|\{v \in V_{G'} : \mathcal{D}(v) \geq d\}| \geq |\{w \in V_H : \mathrm{dist}_H(w,p) \geq d\}|\,.$$

Now let $d \geq \ell$. Then we only have to consider vertices in $F_1 \cup F_2$. Hence, for every $d \geq \ell$:

$$\left|\left\{t_w^i \in V_{F_1} \cup V_{F_2} : \mathcal{D}(t_w^i) \geq d\right\}\right| \geq |\{w \in V_H : \mathrm{dist}_H(w,p) \geq d\}|\,.$$

By construction, for any t_w^i with $i > 1$ we have that $w \in \{s,t\}$ and thus $\mathcal{D}(t_w^i) \leq 1 < \ell \leq d$. Therefore, no vertex t_w^i with $i \neq 1$ is involved in the equation above, so we can write:

$$\left|\left\{t_w^1 \in V_{F_1} \cup V_{F_2} : \mathcal{D}(t_w^1) \geq d\right\}\right| \geq |\{w \in V_H : \mathrm{dist}_H(w,p) \geq d\}|\,.$$

Hence Claim 5.2 is proven.

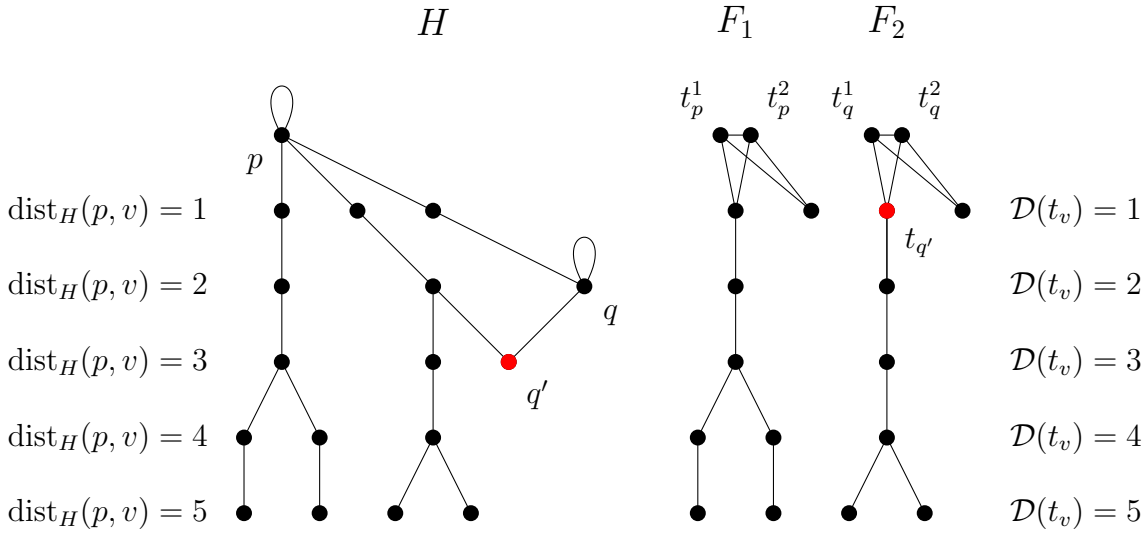


Figure 5.6: An example of a graph H with corresponding graphs F_1 and F_2. Vertices in H equidistant from p are plotted at the same vertical position and likewise vertices $t_v \in F_1$ and $t_w \in F_2$ with $\mathcal{D}(t_v) = \mathcal{D}(t_w)$ are plotted at the same vertical position. The vertices $q' \in H$ and corresponding $t_{q'} \in F_2$ are highlighted.

We first present the intuition behind the final part of the proof. Consider the graphs F_1, F_2 and H in the example shown in Figure 5.6. We recall that every

vertex v (other than p or q) has a single corresponding vertex t_v in F_1 or F_2. We may naturally want to map the vertices of F_1 onto the vertices of H_1, which is possible by definition of F_1. However, when we try to map the vertices of F_2 onto the vertices of H_2, with $h(t_q^i) = p$ (for some i), we will prove that there is at least one vertex q' in H_2 which is further from p in H than it is from q and that cannot be mapped to and thus violates the surjectivity constraint. In Figure 5.6 this vertex, which will play a special role in our proof, is shown in red. In the this figure we observe that there are ten vertices in H (including q') with $\text{dist}_H(p, v) \geq 3$ but only nine vertices (excluding $t_{q'}$) in $F_1 \cup F_2$ with $\mathcal{D}(t_v) \geq 3$ which could be mapped to these vertices. This contradicts Claim 5.2.

We now formally prove that our initial assumption that $p \in h(C_u)$ for all $u \in V_G$ contradicts Claim 5.2. For every vertex x in H_1 there is a corresponding vertex t_x^1 such that $\mathcal{D}(t_x^1) = \text{dist}_{F_1}(t_x^1, t_p) = \text{dist}_{H_1}(x, p)$, where the latter equality follows from the construction of F_1. From Lemma 5.2 we find that $\text{dist}_{H_1}(x, p) = \text{dist}_H(x, p)$ for every $x \in V_{H_1}$. Hence $\mathcal{D}(t_x^1) = \text{dist}_H(x, p)$, and for all $d \geq 0$:

$$\left|\left\{t_x^1 \in V_{F_1} : \mathcal{D}(t_x^1) \geq d\right\}\right| = \left|\{x \in V_{H_1} : \text{dist}_H(x, p) \geq d\}\right|. \tag{5.2}$$

Now let $x \in V_{H_2}$. Using the same arguments, we see that $\mathcal{D}(t_x^1) = \text{dist}_H(x, q)$, and thus $\mathcal{D}(t_x^1) = \text{dist}_H(x, q) \leq \text{dist}_H(x, p)$ by definition. Note that, had we instead supposed that it was q to which everything mapped, we would instead have a strict inequality. As it turns out, we only need the weaker inequality.

We now look for a vertex q' in H_2, such that q' is as far from p as possible, subject to the condition that $\text{dist}_H(q', q) < \text{dist}_H(q', p)$. Let $j = \text{dist}_H(q', p)$. We see that for any vertex x in H_2 such that $\text{dist}_H(x, p) > j$, it is the case that $\text{dist}_H(x, q) = \text{dist}_H(x, p)$. Note that there may be no vertices with $\text{dist}_H(x, q) = \text{dist}_H(x, p)$ in

which case q' is simply the farthest vertex from p within H_2. We also observe that $q' = q$ is possible. So j is well defined and, in fact, we have that $j \geq \ell$.

We now consider the mapping of vertices in H_2 at a distance $d \geq \ell$ from p. We recall that $\mathcal{D}(t^1_x) = \text{dist}_H(x, q)$ for every x in H_2 and that for a vertex $x \in H_2$ of distance at least $j + 1$ from q in H, it holds that $\text{dist}_H(x, q) = \text{dist}_H(x, p)$. Combining this with equation (5.2) yields that:

$$\left|\left\{t^1_x \in V_{F_1} \cup V_{F_2} : \mathcal{D}(t^1_x) > j\right\}\right| = \left|\{x \in V_H : \text{dist}_H(x, p) > j\}\right|. \tag{5.3}$$

However, for $d = j$ we find that, in addition to vertices in H_2 equidistant from p and q, there is at least one vertex that is closer to q than p, namely q', for which it holds that $\mathcal{D}(t^1_{q'}) = \text{dist}_H(q', q) < \text{dist}_H(q', p) = j$. It therefore follows that there are fewer vertices t^1_x with $\mathcal{D}(t^1_x) = j$ than there are vertices x with $\text{dist}_H(x, p) = j$ and hence we see that:

$$\left|\left\{t^1_x \in V_{F_1} \cup V_{F_2} : \mathcal{D}(t^1_x) = j\right\}\right| < \left|\{x \in V_H : \text{dist}_H(x, p) = j\}\right|. \tag{5.4}$$

By combining equations (5.3) and (5.4), we see that:

$$\left|\left\{t^1_x \in V_{F_1} \cup V_{F_2} : \mathcal{D}(t^1_x) \geq j\right\}\right| < \left|\{x \in V_H : \text{dist}_H(x, p) \geq j\}\right|.$$

As $j \geq \ell$, this contradicts Claim 5.2 and concludes the proof of Lemma 5.4. □

We are now ready to state our main result.

Theorem 5.2. *For every connected 2-reflexive graph H, the* Surjective H-Colouring *problem is* NP*-complete.*

Proof. Let H be a connected 2-reflexive graph with reflexive vertices p and q at distance $\ell \geq 2$ from each other. Let ω be the size of a largest clique in H. We

define the graphs H_1, H_2, F_1 and F_2, sets N_p an N_q, and values r_p, r_q as above. Recall that the problem is readily seen to be in NP and that we reduce from (r_p, r_q)-Factor Cut with Roots. From F_1, F_2 and an instance (G, s, t) of the latter problem we construct the graph G'. We claim that G has an (r_p, r_q)-factor cut (V_1, V_2) if and only if there exists a surjective homomorphism h from G' to H.

First suppose that G has an (r_p, r_q)-factor cut (V_1, V_2). By definition, $s \in V_1$ and $t \in V_2$. We define a homomorphism h as follows. For every $x \in V_{F_1} \cup V_{F_2}$, we let h map t^1_x to x. This shows that h is surjective. It remains to define h on the other vertices. For every $u \in V_G$, let h map all of C_u to p if u is in V_1 and let h map all of C_u to q if u is in V_2 (note that this is consistent with how we defined h so far). For each $uv \in E_G$ with $u, v \in V_1$, we map the vertices of the paths P^1_e and P^2_e to p. For each $uv \in E_G$ with $u, v \in V_2$, we map the vertices of the paths P^1_e and P^2_e to q. We are left to show that the vertices of the remaining paths P^1_e and P^2_e can be mapped to appropriate vertices of H.

Note that the red neighbours of each C_u form a clique (whereas all blue vertices of each C_u form an independent set and inner vertices of paths P^1_e and P^2_e have degree 2). However, as (V_1, V_2) is an (r_p, r_q)-factor cut of G, all but at most r_p vertices of these red cliques have been mapped to p already if $u \in V_1$ and all but at most r_q vertices have been mapped to q already if $u \in V_2$. By definition of r_p and r_q, this means that we can map the vertices of the paths P^1_e and P^2_e with $e = uv$ for $u \in V_1$ and $v \in V_2$ to vertices of appropriate shortest paths between p and q in H, so that h is a homomorphism from G' to H (recall that we already showed surjectivity). In particular, the clique formed by the red neighbours of each C_u is mapped to a clique in $N_p \cup \{p\}$ or $N_q \cup \{q\}$.

Now suppose that there exists a surjective homomorphism h from G' to H. For a clique C_u, we may choose any edge e incident to u, such that $C'_u = C_u \cup \{g^{\mathrm{r}}_{u,e}\}$ is a clique of size $\omega + 1$. Since H contains no cliques larger than ω, we find that h maps each clique C'_u (which has size $\omega + 1$) to a clique in H that

contains a reflexive vertex. Note that at least two vertices of C'_u are mapped to a reflexive vertex. Hence we can define the following partition of V_G. We let $V_1 = \{v \in V_G : p \in h(C_v)\}$ and $V_2 = V_G \setminus V_1 = \{v \in V_G : q \in h(C_v)\}$. Lemma 5.4 tells us that $V_1 \neq \emptyset$ and $V_2 \neq \emptyset$. We define $M = \{uv \in E_G : u \in V_1,\, v \in V_2\}$.

Let $e = uv$ be an arbitrary edge in M. By definition, h maps all of C_u to a clique containing p and all of C_v to a clique containing q. Hence, the vertices of the two paths P_e^1 and P_e^2 must be mapped to the vertices of a shortest path between p and q. At most r_p red neighbours of every C_u with $u \in V_1$ can be mapped to a vertex other than p. This is because these red neighbours form a clique. As such they must be mapped onto vertices that form a clique in H. As such vertices lie on a shortest path from p to q, the clique in H has size at most r_p. Similarly, at most r_q red neighbours of every C_u with $u \in V_2$ can be mapped to a vertex other than q. As such, (V_1, V_2) is an (r_p, r_q)-factor cut in G. □

A Small Extension. Two vertices u and v in a graph G are *true twins* if they are adjacent to each other and share the same neighbours in $V_G \setminus \{u, v\}$. Let $H^{(i,j)}$ be a graph obtained from a connected 2-reflexive graph H with reflexive vertices p and q after introducing i reflexive true twins of p and j reflexive true twins of q. In the graph G' we increase the cliques C_u to size $\omega + \max(i, j)$. We call the resulting graph G''. Then it is readily seen that there exists a surjective homomorphism from G' to H if and only if there exists a surjective homomorphism from G'' to $H^{(i,j)}$.

Since creating true twins of an irreflexive vertex will not prevent a graph from being 2-reflexive, we can state the following result.

Theorem 5.3. *For any graph H that can be obtained from a 2-reflexive graph H' by replacing vertices with true twins; the* Surjective H-Colouring *problem is* NP*-complete.*

5.3 Target Graphs Of At Most Four Vertices

In this section we classify the computational complexity of Surjective H-Colouring for every target graph H with at most four vertices. We require a number of lemmas. The first lemma is proved for compaction and not vertex-surjection. However, the only property of compaction used is vertex-surjection and so it is easy to see it holds in this modified form. The second lemma is also displayed in Figure 5.2.

Lemma 5.5 ([96]). *Let H be a graph with connected components $H_1, \ldots, H_s$. If* Surjective H_i-Colouring *is* NP*-complete for some i, then* Surjective H-Colouring *is also* NP*-complete.*

Lemma 5.6 ([6]). *For every graph H, if* H-Compaction *is polynomial-time solvable, then* Surjective H-Colouring *is polynomial-time solvable.*

We also need two results of Golovach, Paulusma and Song. Recall that in our context a tree is a connected graph with no cycles of length at least 3.

Lemma 5.7 ([46]). *Let H be an irreflexive non-bipartite graph. Then* Surjective H-Colouring *is* NP*-complete.*

Lemma 5.8 ([46]). *Let H be a tree. Then* Surjective H-Colouring *is solvable in polynomial time if H is loop-connected and* NP*-complete otherwise.*

Recall that C_4^* denotes the reflexive cycle on four vertices (see Figure 5.7 for an example).

Lemma 5.9 ([76]). *The* Surjective C_4^*-Colouring *problem is* NP*-complete.*

We let D denote the irreflexive diamond, that is, the irreflexive complete graph on four vertices minus an edge. The (irreflexive) paw is the graph obtained from the triangle after attaching a pendant vertex to one of the vertices of the triangle,

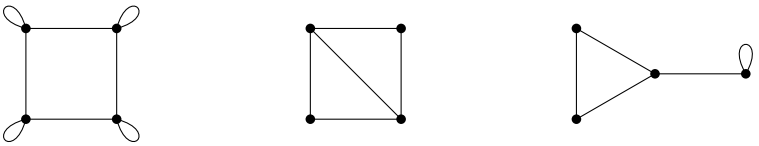


Figure 5.7: The graphs C_4^*, D and paw*.

that is, the graph with vertices x_1, x_2, y, z and edges x_1x_2, x_1y, x_2y, yz. We let paw* denote the graph obtained from the paw after adding a loop to its vertex of degree 1 (that is, following the above notation, the loop zz). Both D and paw* are displayed in Figure 5.7 as well.

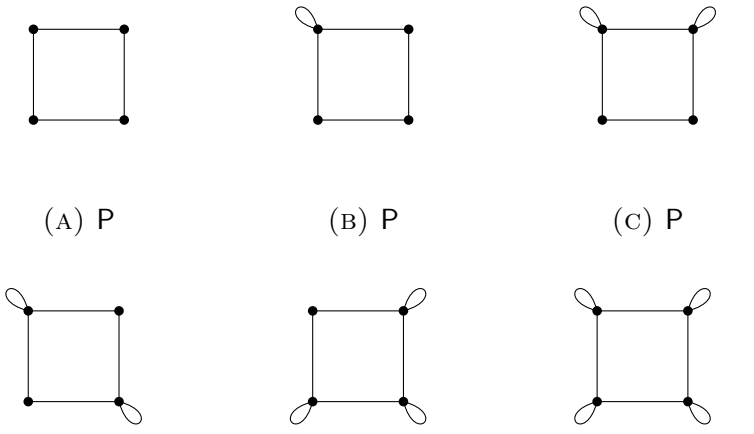
We are now ready to state our main result.

Theorem 5.4. *Let H be a graph with $|V_H| \leq 4$. Then* Surjective H-Colouring *is* NP*-complete if some connected component of H is not loop-connected or is an irreflexive complete graph on at least three vertices, or $H \in \{C_4^*, D, \text{paw}^*\}$. Otherwise* Surjective H-Colouring *is polynomial-time solvable.*

Proof. Let H be a graph on at most four vertices. If H is a loop-connected forest (that is, every component of H is loop-connected) or H has a dominating reflexive vertex, then Vikas [96] showed that H-Compaction is in P. Hence, Surjective H-Colouring is in P by Lemma 5.6. If H contains a component that is a non-loop-connected tree, then Surjective H-Colouring is NP-complete by Lemmas 5.5 and 5.8. If H is an irreflexive non-bipartite graph, then Surjective H-Colouring is NP-complete by Lemma 5.7.

Note that the above cases cover all graphs H on at most three vertices, all disconnected graphs H on four vertices and all trees H on four vertices. The only two graphs H on at most three vertices for which Surjective H-Colouring is NP-complete are the irreflexive cycle on three vertices and the 3-vertex path in which the two end-vertices are reflexive. The only disconnected graphs H on four vertices for which Surjective H-Colouring is NP-complete are those that

contain these two graphs as connected components. The only trees H on four vertices for which Surjective H-Colouring is NP-complete are those that are not loop-connected. Hence the theorem holds for every graph H on at most three vertices, for every disconnected graph H on four vertices and for every tree H on four vertices.



(a) P (b) P (c) P

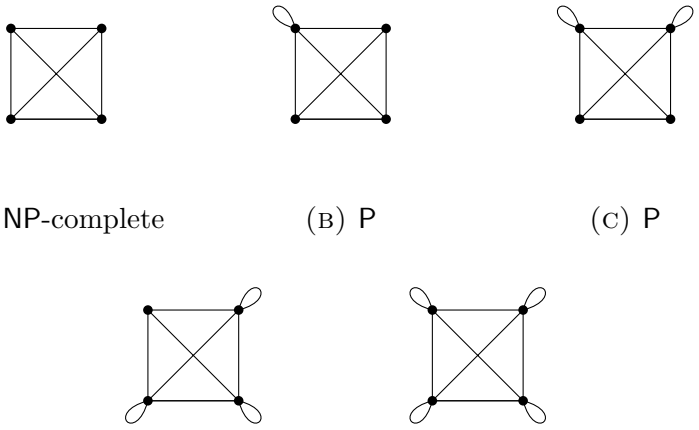
(d) NP-complete (e) P (f) NP-complete

Figure 5.8: All cycles H on four vertices.

From now on we assume that H is a connected graph on four vertices that is not a tree. Then H is either the cycle on four vertices, the complete graph on four vertices, the diamond or the paw. We consider each of these cases separately.

Suppose H is the cycle on four vertices. There are six cases to consider (see Figure 5.8). If H is reflexive, then Surjective H-Colouring is NP-complete by Lemma 5.9. If H is not loop-connected, then H is 2-reflexive, and thus Surjective H-Colouring is NP-complete by Theorem 5.2. In the remaining four cases H is loop-connected. For each of these target graphs, Vikas [96] showed that H-Compaction is in P. Hence, Surjective H-Colouring is in P by Lemma 5.6. We find that the theorem holds when H is a cycle on four vertices.

Suppose H is the complete graph on four vertices. There are five cases to consider (see Figure 5.9). If H is irreflexive, then Surjective H-Colouring is



(a) NP-complete (b) P (c) P

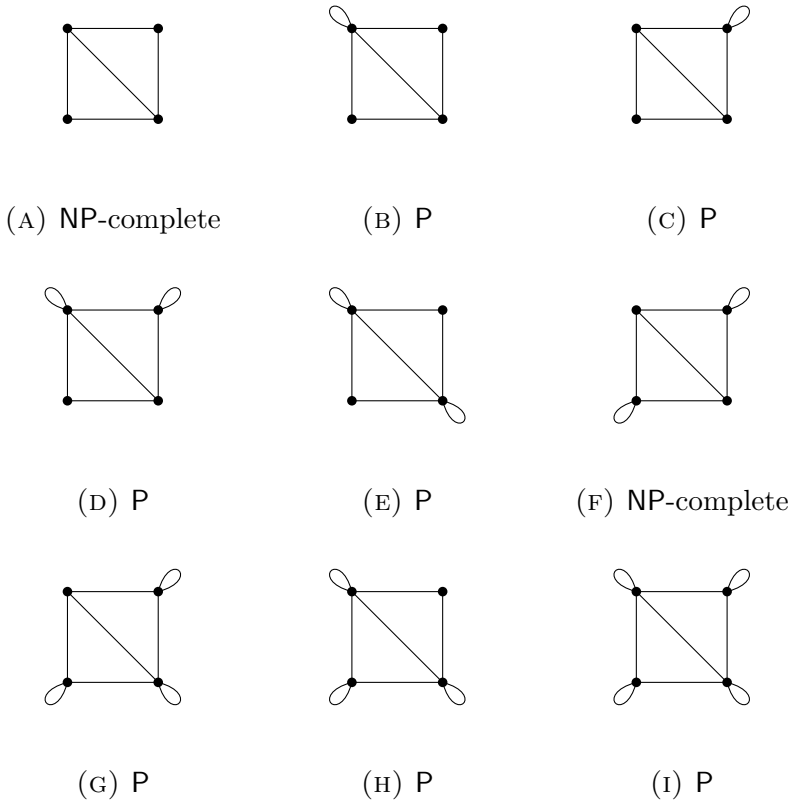
(d) P (e) P

Figure 5.9: All complete graphs H on four vertices.

NP-complete by Lemma 5.7 (as H is non-bipartite as well). For each of the other four target graphs, Vikas [96] showed that H-Compaction is in P. Hence, Surjective H-Colouring is in P by Lemma 5.6. We find that the theorem holds when H is the complete graph on four vertices.

Suppose H is the diamond. There are nine cases to consider (see Figure 5.10 on page 115). If H is irreflexive, then Surjective H-Colouring is NP-complete by Lemma 5.7 (as H is non-bipartite as well). If H is not loop-connected, then H is 2-reflexive, and thus Surjective H-Colouring is NP-complete by Theorem 5.2. For the remaining seven target graphs, Vikas [96] showed that H-Compaction is in P. Hence, Surjective H-Colouring is in P by Lemma 5.6. We find that the theorem holds when H is the diamond.

Suppose H is the paw with vertices x_1, x_2, y, z and edges x_1x_2, x_1y, x_2y and yz and possibly one or more loops. There are twelve cases to consider (see Figure 5.11 on page 116). If H is irreflexive, then Surjective H-Colouring is NP-complete by Lemma 5.7 (as H is non-bipartite as well). If H is not loop-connected, then the set of reflexive vertices is formed by one or two vertices from $\{x_1, x_2\}$ and z. Then Surjective H-Colouring is NP-complete by Theorem 5.3. We are left with nine cases. Vikas [96] showed that H-Compaction is in P for all of these



(a) NP-complete (b) P (c) P

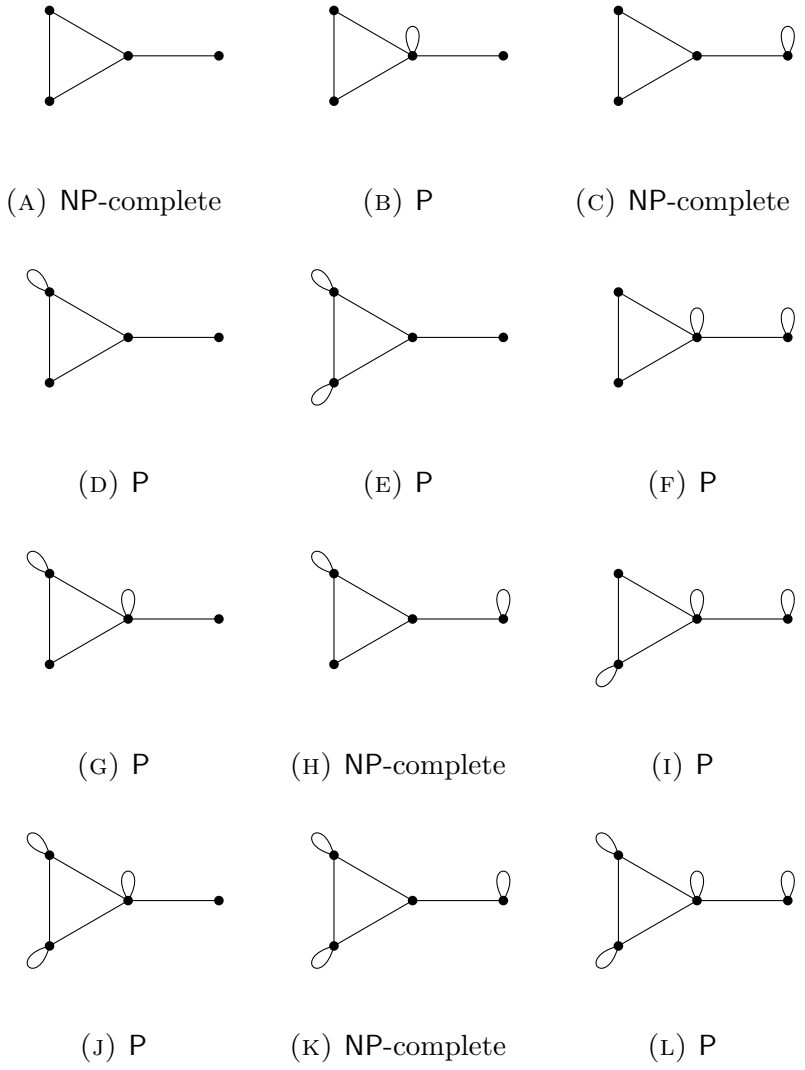
(d) P (e) P (f) NP-complete

(g) P (h) P (i) P

Figure 5.10: All diamonds H on four vertices.

cases except for the case where z is the only reflexive vertex. Hence, for eight of these nine cases, Surjective H-Colouring is in P by Lemma 5.6.

We are left to consider the case in which z is the (only) reflexive vertex. Recall that we denote this target graph by paw*. Theorem 3.5 of [96] proves that paw*-Compaction is NP-complete using a reduction from C_3-Retraction (which is NP-complete), but we will argue the proof also works in the case of Surjective paw*-Colouring. It is shown that (i) a graph G retracts to C_3 if and only if a certain graph G' retracts to paw* if and only if (iii) G' compacts to paw*. The salient part of the proof is Lemma 3.5.2 of [96], in which it is argued that (ii) and (iii) are equivalent. We note that if a graph retracts to another graph, then there exists a surjective homomorphism from the first graph to the second graph. Hence, we need to verify only whether G' retracts to paw* should there exist a surjective homomorphism from G' to paw*. In the proof of Lemma 3.5.2 of [96], the properties of compaction are only used three times. The first two



(a) NP-complete (b) P (c) NP-complete

(d) P (e) P (f) P

(g) P (h) NP-complete (i) P

(j) P (k) NP-complete (l) P

Figure 5.11: All paws H on four vertices.

are paragraph 2, line 2 and paragraph 7, line 4 (in the proof of Lemma 3.5.2). The only property used of compaction on these two occasions is vertex surjection. Finally, compaction is alluded to in the final paragraph of the proof, but here any homomorphism would have the desired property. Thus, Vikas [96] has actually proved that G' retracts to paw* if and only if G' has a surjective homomorphism to paw*, and it follows that Surjective paw*-Colouring is NP-complete.

From the above we conclude that the theorem holds in all cases when H is the paw. This completes the proof of Theorem 5.4. □

From the proof of Theorem 5.4 it follows that whenever H is a target graph on at most four vertices for which H-Compaction is polynomial-time solvable, then so is Surjective H-Colouring. Vikas [96] also showed that for every target graph H on at most four vertices for which Surjective H-Colouring is NP-complete, H-Compaction is NP-complete. Hence, Theorem 5.4 corresponds to Vikas' complexity classification of H-Compaction for targets graphs H of at most four vertices. We also refer to Vikas [96] for the complexity equivalence of H-Compaction and H-Retraction. Thus, we obtained the following corollary.

Corollary 5.1. *Let H be a graph on at most four vertices. Then the three problems* Surjective H-Colouring*, H-*Compaction *and H-*Retraction *are polynomially equivalent.*

5.4 Conclusions

There are two main challenges in the area of Surjective H-Colouring. The first would be a complete dichotomy of Surjective H-Colouring between P and NP-complete, however, this still seems to be difficult. A first goal could be to prove that Surjective H-Colouring is NP-complete for every connected graph H that is not loop-connected. However, doing this using our current techniques does not seem straightforward and we may need new hardness reductions. In this chapter we have made another step towards this by showing that Surjective H-Colouring is NP-complete for every connected, 2-reflexive graph H that is not loop-connected. The next step along this avenue would be to try and extend the result to k-reflexive graphs, however, this has proven to be very complex.

The second challenge is to prove polynomial equivalence between the three problems Surjective H-Colouring, H-Compaction and H-Retraction. However, completely achieving this goal also seems far from trivial. Our classification for target graphs H up to four vertices does show such an equivalence for these cases.

Bibliography

[1] M. H. Alsuwaiyel. *Algorithms: Design Techniques and Analysis*. World Scientific, 1999.

[2] K. Appel and W Haken. Every planar map is four colorable. Part I: Discharging. *Illinois Journal of Mathematics*, 21:429–490, 1977.

[3] A. Atminas, V. V. Lozin, and I. Razgon. Linear time algorithm for computing a small biclique in graphs without long induced paths. *Proc. SWAT 2012, Lecture Notes in Computer Science*, 7357:142–152, 2012.

[4] R. Beigel and D. Eppstein. 3-Coloring in time $\mathcal{O}(1.3289^n)$. *Journal of Algorithms*, 54(2):168–204, 2005.

[5] N. Biggs, E. K. Lloyd, and R. J. Wilson. *Graph Theory, 1736-1936*. Clarendon Press, New York, NY, USA, 1986.

[6] M. Bodirsky, J. Kára, and B. Martin. The complexity of surjective homomorphism problems – a survey. *Discrete Applied Mathematics*, 160:1680–1690, 2012.

[7] H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25:1305–1317, 1996.

[8] P. S. Bonsma. The complexity of the matching-cut problem for planar graphs and other graph classes. *Journal of Graph Theory*, 62:109–126, 2009.

[9] A. Brandstädt, F. F. Dragan, V. B. Le, and T. Szymczak. On stable cutsets in graphs. *Discrete Applied Mathematics*, 105:39–50, 2000.

[10] A. Brandstädt, V. B. Le, and J. Spinrad. Graph classes: A survey. *SIAM Monographs on Discrete Mathematics and Applications, Society for Industrial and Applied Mathematics (SIAM)*, 1999.

[11] H. J. Broersma, F. V. Fomin, R. Královič, and G.J. Woeginger. Eliminating graphs by means of parallel knock-out schemes. *Discrete Applied Mathematics*, 155:92–102, 2007.

[12] H. J. Broersma, M. Johnson, and D. Paulusma. Upper bounds and algorithms for parallel knock-out numbers. *Theoretical Computer Science*, 410:1329–1327, 2008.

[13] H. J. Broersma, M. Johnson, D. Paulusma, and I. A. Stewart. The computational complexity of the parallel knock-out problem. *Theoretical Computer Science*, 393:182–195, 2008.

[14] K. Cameron, E. M. Eschen, C. T. Hoàng, and R. Sritharan. The complexity of the list partition problem for graphs. *SIAM Journal on Discrete Mathematics*, 21:900–929, 2007.

[15] V. Chvátal. Recognizing decomposable graphs. *Journal of Graph Theory*, 8:51–53, 1984.

[16] M. Cochefert, J-F. Couturier, P. A. Golovach, D. Kratsch, and D. Paulusma. Sparse square roots. *Proc. WG 2013, LNCS*, 8165:177–188, 2013.

[17] M. Cochefert, J-F. Couturier, P. A. Golovach, D. Kratsch, and D. Paulusma. Parameterized algorithms for finding square roots. *Algorithmica*, 74:602–629, 2016.

[18] D. G. Corneil, H. Lerchs, and L. Stewart Burlingham. Complement reducible graphs. *Discrete Applied Mathematics*, 3:163–174, 1981.

[19] D. G. Corneil, Y. Perl, and L. K. Stewart. A linear recognition algorithm for cographs. *SIAM Journal on Computing*, 14:926–934, 1985.

[20] B. Courcelle. The monadic second-order logic of graphs. I. recognizable sets of finite graphs. *Information and Computation*, 85:12–75, 1990.

[21] B. Courcelle. The monadic second-order logic of graphs. III. tree-decompositions, minor and complexity issues. *Informatique Théorique et Applications*, 26:257–286, 1992.

[22] B. Courcelle and S. Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101:77–144, 2000.

[23] W. H. Cunnigham and J. Edmonds. A combinatorial decomposition theory. *Canadian Journal of Mathematics*, 32:734–765, 1980.

[24] C. M. H. de Figueiredo, S. Klein, and B. Reed. Finding skew partitions efficiently. *Journal of Algorithms*, 37:505–521, 2000.

[25] H. N. de Ridder et al. Information System on Graph Classes and their Inclusions, 2001.

[26] R. Diestel. *Graph Theory, 4th Edition, Graduate Texts in Mathematics, vol. 173*. Springer, 2012.

[27] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer-Verlag Monographs in Computer Science, 1999.

[28] B. Esfahbod. Euler diagram for P, NP, NP-complete, and NP-hard set of problems, 2007.

[29] W. Espelage, F. Gurski, and E. Wanke. How to solve np-hard graph problems on clique-width bounded graphs in polynomial time. *Proc. WG 2001, Lecture Notes in Computer Science*, 2204:117–128, 2001.

[30] L. Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum imperialis Petropolitanae*, 8:128–140, 1741.

[31] B. Farzad and M. Karimi. Square-root finding problem in graphs, a complete dichotomy theorem. *CoRR*, abs/1210.7684, 2012.

[32] B. Farzad, L. C. Lau, V. B. Le, and N. N. Tuy. Complexity of finding graph roots with girth conditions. *Algorithmica*, 62:38–53, 2012.

[33] T. Feder, P. Hell, and J. Huang. Bi-arc graphs and the complexity of list homomorphisms. *Journal of Graph Theory*, 42:61–80, 2003.

[34] T. Feder, P. Hell, P. Jonsson, A. Krokhin, and G. Nordh. Retractions to pseudoforests. *SIAM Journal on Discrete Mathematics*, 24:101–112, 2010.

[35] T. Feder and M. Y. Vardi. The computational structure of monotone monadic snp and constraint satisfaction: a study through datalog and group theory. *SIAM Journal on Computing*, 28:57–104, 1998.

[36] J. Fiala and J. Kratochvíl. Locally constrained graph homomorphisms – structure, complexity, and applications. *Computer Science Review*, 2:97–111, 2008.

[37] J. Fiala and D. Paulusma. A complete complexity classification of the role assignment problem. *Theoretical Computer Science*, 349:67–81, 2005.

[38] S. Földes and P. L. Hammer. Split graphs. *8th South–Eastern Conf. on Combinatorics, Graph Theory and Computing, Congressus Numerantium*, 19:311–315, 1977.

[39] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.

[40] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete problems. *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, STOC'74:47–63, 1974.

[41] P. A. Golovach, M. Johnson, B. Martin, D. Paulusma, and A. Stewart. Surjective h-Colouring: New hardness results. *CoRR*, abs/1701.02188, 2017.

[42] P. A. Golovach, D. Kratsch, D. Paulusma, and A. Stewart. Finding cactus roots in polynomial time. *Proc. IWOCA 16, Lecture Notes in Computer Science*, 9843:361–372, 2016.

[43] P. A. Golovach, D. Kratsch, D. Paulusma, and A. Stewart. A linear kernel for finding square roots of almost planar graphs. *Proc. SWAT 16, Leibniz International Proceedings in Informatics*, 53:4:1–4:14, 2016.

[44] P. A. Golovach, D. Kratsch, D. Paulusma, and A. Stewart. Squares of low clique number. *Proc. CTW 16, Electronic Notes in Discrete Mathematics*, 55:195–198, 2016.

[45] P. A. Golovach, B. Lidický, B. Martin, and D. Paulusma. Finding vertex-surjective graph homomorphisms. *Acta Informatica*, 49:381–394, 2012.

[46] P. A. Golovach, D. Paulusma, and J. Song. Computing vertex-surjective homomorphisms to partially reflexive trees. *Theoretical Computer Science*, 457:86–100, 2012.

[47] M. C. Golumbic and U. Rotics. On the clique-width of some perfect graph classes. *International Journal of Foundations of Computer Science*, 11:423–443, 2000.

[48] M. Grohe, K. Kawarabayashi, D. Marx, and P. Wollan. Finding topological subgraphs is fixed-parameter tractable. *Proc. STOC*, 2011:479–488, 2011.

[49] P. Hammer and F. Maffray. Completely separable graphs. *Discrete Applied Mathematics*, 27:85–99, 1990.

[50] F. Harary, R. M. Karp, and W. T. Tutte. A criterion for planarity of the square of a graph. *Journal of Combinatorial Theory*, 2:395–405, 1967.

[51] P. Heggernes, P. van 't Hof, D. Marx, N. Misra, and Y. Villanger. On the parameterized complexity of finding separators with non-hereditary properties. *Algorithmica*, 72:687–713, 2015.

[52] P. Hell and J. Nešetřil. On the complexity of h-colouring. *Journal of Combinatorial Theory. Series B*, 48:92–110, 1990.

[53] P. Hell and J. Nešetřil. *Graphs and Homomorphisms*. Oxford University Press, 2004.

[54] T. Ito, M. Kamiński, D. Paulusma, and D. M. Thilikos. On disconnected cuts and separators. *Discrete Applied Mathematics*, 159:1345–1351, 2011.

[55] T. Ito, M. Kamiński, D. Paulusma, and D. M. Thilikos. Parameterizing cut sets in a graph by the number of their components. *Theoretical Computer Science*, 412:6340–6350, 2011.

[56] O. Johansson. Clique-decomposition. *NLC-decomposition, and modular decomposition – relationships and results for random graphs, Congressus Numerantium*, 132:39–60, 1998.

[57] M. Johnson, D. Paulusma, and A. Stewart. Knocking out P_k-free graphs. *Proc. MFCS 2014, Lecture Notes in Computer Science*, 8635:396–407, 2014.

[58] M. Johnson, D. Paulusma, and A. Stewart. Knocking out P_k-free graphs. *Discrete Applied Mathematics*, 190:100–108, 2015.

[59] M. Johnson, D. Paulusma, and C. Wood. Path factors and parallel knock-out schemes of almost claw-free graphs. *Discrete Mathematics*, 310:1413–1423, 2010.

[60] M. Kamiński, D. Paulusma, A. Stewart, and D. M. Thilikos. Minimal disconnected cuts in planar graphs. *Proc. FCT 2015, LNCS*, 9210:243–254, 2015.

[61] M. Kamiński, D. Paulusma, A. Stewart, and D. M. Thilikos. Minimal disconnected cuts in planar graphs. *Networks*, 68:250–259, 2016.

[62] M. Kamiński, D. Paulusma, and D. M. Thilikos. Contractions of planar graphs in polynomial time. *Proc. ESA 2010, LNCS*, 6346:122–133, 2010.

[63] Richard. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Springer US, 1972.

[64] W. S. Kennedy and B. Reed. Fast skew partition recognition. *In: Computational Geometry and Graph Theory, Lecture Notes in Computer Science*, 4535:101–107, 2008.

[65] D. E. Lampert and P. J. Slater. Parallel knockouts in the complete graph. *American Mathematical Monthly*, 105:556–558, 1998.

[66] L. C. Lau. Bipartite roots of graphs. *ACM Transactions on Algorithms*, 2:178–208, 2006.

[67] L. C. Lau and D. G. Corneil. Recognizing powers of proper interval, split, and chordal graphs. *SIAM Journal on Discrete Mathematics*, 18:83–102, 2004.

[68] V. B. Le, R. Mosca, and H. Müller. On stable cutsets in claw-free graphs and planar graphs. *J. Discrete Algorithms*, 6:256–276, 2008.

[69] V. B. Le, A. Oversberg, and O. Schaudt. Polynomial time recognition of squares of ptolemaic graphs and 3-sun-free split graphs. *Theoretical Computer Science*, 648:26–33, 2015.

[70] V. B. Le, A. Oversberg, and O. Schaudt. A unified approach for recognizing squares of split graphs. *Theoretical Computer Science*, 602:39–49, 2015.

[71] V. B. Le and N. N. Tuy. The square of a block graph. *Discrete Mathematics*, 310:734–741, 2010.

[72] V. B. Le and N. N. Tuy. A good characterization of squares of strongly chordal split graphs. *Information Processing Letters*, 111:120–123, 2011.

[73] Y. Lin and S. Skiena. Algorithms for square roots of graphs. *SIAM Journal on Discrete Mathematics*, 8:99–118, 1995.

[74] Transport For London. Standard tube map, 2016.

[75] Donald MacKenzie. *Mechanizing Proof: Computing, Risk, and Trust*. MIT Press, 2004.

[76] B. Martin and D. Paulusma. The computational complexity of disconnected cut and 2k2-partition. *Journal of Combinatorial Theory, Series B*, 111:17–37, 2015.

[77] D. Marx, B. O'Sullivan, and I. Razgon. Finding small separators in linear time via treewidth reduction. *ACM Transactions on Algorithms*, 9:30:1–30:35, 2013.

[78] M. Milanic, A. Oversberg, and O. Schaudt. A characterization of line graphs that are squares of graphs. *Discrete Applied Mathematics*, 173:83–91, 2014.

[79] M. Milanic and O. Schaudt. Computing square roots of trivially perfect and threshold graphs. *Discrete Applied Mathematics*, 161:1538–1545, 2013.

[80] B. Mohar. A linear time algorithm for embedding graphs in an arbitrary surface. *SIAM Journal on Discrete Mathematics*, 12:6–26, 1999.

[81] B. Mohar and C. Thomassen. *Graphs on Surfaces*. The Johns Hopkins University Press, 2001.

[82] R. Motwani and M. Sudan. Computing roots of graphs is hard. *Discrete Applied Mathematics*, 54:81–88, 1994.

[83] A. Mukhopadhyay. The square root of a graph. *Journal of Combinatorial Theory*, 2:290–295, 1967.

[84] N. V. Nestoridis and D. M. Thilikos. Square roots of minor closed graph classes. *Discrete Applied Mathematics*, 168:34–39, 2014.

[85] M. Patrignani and M. Pizzonia. The complexity of the matching-cut problem. *Proc. WG 2001, Lecture Notes in Computer Science*, 2204:284–295, 2001.

[86] F. P. Ramsey. On a problem of formal logic. *Proceedings of the London Mathematical Society*, 30:264–286, 1930.

[87] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978.

[88] N. Robertson, P. Seymour, and R. Thomas. Quickly excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 62:323–348, 1994.

[89] N. Robertson and P. D. Seymour. Graph minors. xvi. excluding a non-planar graph. *Journal of Combinatorial Theory Series B*, 89:43–76, 2003.

[90] Neil Robertson, Daniel P. Sanders, Paul Seymour, and Robin Thomas. Efficiently four-coloring planar graphs. *Proceedings of the twenty-eighth Annual ACM symposium on Theory of computing*, STOC'96:571–575, 1996.

[91] I. C. Ross and F. Harary. The square of a tree. *Bell System Technical Journal*, 39:641–647, 1960.

[92] R. E. Tarjan. Decomposition by clique separators. *Discrete Math.*, 55:221–232, 1985.

[93] R. Thomas. An update on the four-color theorem. *Notices of the American Mathematical Society*, 45:848–859, 1998.

[94] N. Vikas. Computational complexity of compaction to reflexive cycles. *SIAM Journal on Computing*, 32:253–280, 2002.

[95] N. Vikas. Compaction, Retraction, and Constraint Satisfaction. *SIAM Journal on Computing*, 33:761–782, 2004.

[96] N. Vikas. A complete and equal computational complexity classification of compaction and retraction to all graphs with at most four vertices and some general results. *Journal of Computer and System Sciences*, 71:406–439, 2005.

[97] Narayan Vikas. Algorithms for partition of some class of graphs under compaction and vertex-compaction. *Algorithmica*, 67(2):180–206, 2013.

[98] E. Wanke. k-NLC graphs and polynomial algorithms. *Discrete Applied Mathematics*, 54:251–266, 1994.

[99] S. H. Whitesides. An algorithm for finding clique cut-sets. *Information Processing Letters*, 12:31–32, 1981.

[100] Robin Wilson. *Four Colors Suffice: How the Map Problem Was Solved*. Princeton University Press, 2002.